НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра автоматизованих систем обробки інформації і управління

«На правах рукопису»

«До захисту допущено»
В.о. завідувача кафедри
_____ *О.А.Павлов*
(підпис) (ініціали, прізвище)
"___"_____2019 р.

УДК _____

Магістерська дисертація

зі спеціальності *121 «Інженерія програмного забезпечення»*

на тему: «*Математичне та програмне забезпечення для аналізу потоків текстових даних*»

Виконав: студент VI курсу, групи *ІП-82мп*

Степанюк Євгеній Юрійович _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник *старший викладач Олійник Ю.І.* _____
(посада, науковий ступінь, , вчене звання, прізвище та ініціали) (підпис)

Консультант *доц.,к.т.н. Ліщук К.І.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент *доц.каф. ТК, к.т.н., доц., к.т.н. Лісовиченко О.І.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2019 року

РЕФЕРАТ

Актуальність теми:

а) покращення якості визначення емоцій в повідомленнях написаних українською мовою;

б) ефективний інструмент моніторингу і оцінювання тематики груп/каналів.

Мета дослідження дисертаційної роботи є покращення сентимент аналізу у текстових потоках даних.

Розроблені методи і програмні засоби повинні відповідати таким вимогам:

– більш висока в порівнянні з існуючими моделями якість запропонованих методів;

– адаптація методів для підтримки текстів різних мов. В даній дисертаційній роботі розглядаються тексти користувачів українською та англійською мовами;

– візуалізація результатів аналізу.

Для реалізації поставленої мети були сформульовані **наступні завдання**:

а) провести аналіз існуючих методів та програмних рішень;

б) обґрунтувати вибір методу сентимент аналізу;

в) розробити математичну модель класифікації повідомлень користувачів за сентиментом;

г) покращити методи аналізу емоційної складової для потоків даних;

д) реалізувати запропоновані методи у вигляді програмного засобу. візуалізувати аналіз емоційної складової потоку за певний період даних та провести експериментальні дослідження з метою визначення ефективності роботи методів і моделей.

Об'єкт і предмет дослідження. Об'єктом дослідження є потік повідомлень користувачів, представлені у вигляді потоку неструктурованих текстів на природній мові.

Методи дослідження: У даній дисертаційній роботі застосовувалися методи обробки природної мови, засновані на правилах, словниках та існуючих лінгвістичних ресурсах, і ймовірнісних тематичних моделях, заснованих на комплексі методів машинного навчання.

Наукова новизна: У даній роботі запропоновані поліпшення для методів визначення сентименту в текстах, написаних українською мовою, представлених у вигляді потоку неструктурованих текстових даних природньою мовою, з використання алгоритму машинного навчання з учителем.

Практичне значення отриманих результатів визначається тим, що запропоноване покращення до алгоритму може бути використано для прикладного аналізу емоційної складової контенту текстових каналів, груп в месенджері telegram і т.д..

Зв’язок роботи з науковими програмами, планами, темами: робота виконувалась на кафедрі автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського» в рамках теми «Методи та

технології високопродуктивних обчислень та обробки надвеликих масивів даних». Державний реєстраційний номер 0117U000924.

Публікації: Наукові положення дисертації опубліковані в тезах конференції *«ІНФОРМАТИКА ТА ОБЧИСЛЮВАЛЬНА ТЕХНІКА – ІОТ-2019»*.

Апробація: Основні положення роботи доповідались і обговорювались на конференції *«ІНФОРМАТИКА ТА ОБЧИСЛЮВАЛЬНА ТЕХНІКА – ІОТ-2019»*.

Ключові слова: СЕМАНТИЧНИЙ АНАЛІЗ, ПОТІК ДАНИХ, GRADIENT BOOSTING TREE, АНАЛІЗ ТОНАЛЬНОСТІ

ABSTRACT

The Actuality of theme:

a) improving the quality of the definition of emotions in messages written in Ukrainian;

b) an effective tool for monitoring and evaluating group / channel topics.

The purpose of the dissertation research is to improve sentiment analysis in text data streams.

The methods and software developed must meet the **following requirements**:

- the quality of the proposed methods is higher than the existing models;

- adaptation of methods to support texts in different languages. This dissertation deals with the texts of users in Ukrainian and English;

- visualization of analysis results.

To achieve this goal, the following **tasks were formulated**:

a) to analyze existing methods and software solutions;

b) justify the choice of sentiment analysis method;

c) to develop a mathematical model for the classification of user messages by sentiment;

d) develop a method of analysis of the emotional component for streaming data;

e) to implement the proposed methods in the form of software. visualize the analysis of the emotional component of the flow over a given period of data and conduct experimental studies to determine the effectiveness of methods and models.

Object and object of research. The object of the study is the stream of user messages, represented as a stream of unstructured text in natural language.

Research Methods: In this dissertation, natural language processing methods based on rules, dictionaries and existing linguistic resources, and probabilistic thematic models based on a set of machine learning methods have been applied.

Scientific Novelty: This paper proposes improvements for methods of determining sentiment in texts written in Ukrainian, presented as a flow of unstructured textual data in natural language, using a machine learning algorithm with a teacher.

The practical significance of the obtained results is determined by the fact that the proposed enhancement to the algorithm can be used for applied analysis of the emotional content of the content of channels, groups, etc. in the telegram manager.

Relationship with working with scientific programs, plans, topics: work was performed at the Department of Automated Information Processing and Management Systems of the National Technical University of Ukraine “Kyiv Polytechnic Institute. Igor Sikorsky” within the topic “Methods and technologies of high-performance computing and processing of large data sets ”. State Registration Number 0117U000924.

Publications: Theses of the thesis are published in *«ІНФОРМАТИКА ТА ОБЧИСЛЮВАЛЬНА ТЕХНІКА – ІОТ-2019»*.

Testing: The main provisions of the work were reported and discussed at the conference "Informatics and Computer Engineering - IOT-2019".

Keywords: SEMANTIC ANALYSIS, DATA FLOW, GRADIENT BOOSTING TREE, TONALYSIS ANALYSIS

ЗМІСТ

ВСТУП

Сентимент-аналіз інформаційних потоків має великий потенціал застосування для моніторингових, аналітичних і сигнальних систем, для систем документообігу і рекламних платформ, націлених по тематиці веб-сторінок. Текст природною мовою, крім інформації, може висловлювати емоційну оцінку того, про що повідомляється.

Історично склалося так, що традиційний підхід до сентимент аналізу являє собою задачу класифікації тексту (частини тексту) на дві-три категорії (негативний, позитивний, нейтральний або просто: негативний або позитивний). Саме з такого завдання почав свій розвиток аналіз тональності текстів, відгуків користувачів та ін.

Для рішення такого роду задач існує достатня кількість підходів та алгоритмів, що показують гарні результати при практичному застосуванні. Проте стандартні підходи не завжди можуть дати гарний результат коли мова піде про обробку потокових даних.

В останні десятиліття на ринку з'явилось дуже багато месенджерів, форумів де люди спілкуються між собою. Це пов'язано, перш за все, з розвитком інтернету та його доступністю для людей, що призводить до постійного збільшення кількості активних користувачів на форумах та в групах в месенджерах. У зв'язку з цим у адміністраторів виникає проблема підтримання контролю у них, визначення негативних чи позитивних емоцій в повідомленнях, що б відповідним чином реагувати. Також як приклад ще однієї прикладної задачі може слугувати визначення тональності повідомлень або публікацій в соціальних мережах, коли ця тональність різко змінюється. Таким чином постає проблема автоматичного аналізу емоцій в таких потоках даних, оскільки кількість користувачів активно збільшується.

Дана магістерська дисертація присвячена модернізації та пошуку рішення для можливості аналізу тональності текстових потоків даних. Доцільність дослідження обґрунтовано на прикладі прикладних задач про визначення зміни тональності текстових повідомлень в соціальних мережах або на форумах.

Метою дисертаційної роботи є покращення сентимент аналізу у текстових потоках даних.

Розроблені методи і програмні засоби повинні відповідати таким вимогам:

– більш висока в порівнянні з існуючими моделями якість запропонованих методів;

– адаптація методів для підтримки текстів різних мов. В даній дисертаційній роботі розглядаються тексти користувачів українською та англійською мовами;

– візуалізація результатів аналізу.

Об'єкт і предмет дослідження. Об'єктом дослідження є потік повідомлень користувачів, представлені у вигляді потоку неструктурованих текстів на природній мові.

Для досягнення поставленої мети необхідно було вирішити такі **завдання**:

е) провести аналіз інстуючих методів та програмних рішень;

ж) обгрунтувати вибір методу сентимент аналізу;

з) розробити математичну модель класифікації повідомлень користувачів за сентиментом;

и) розробити метод аналізу емоційної складової для потокових даних;

к) реалізувати запропоновані методи у вигляді програмного засобу. Візуалізовувати аналіз емоційної складової потоку за певний період даних та провести експериментальні дослідження з метою визначення ефективності роботи методів і моделей.

Методи досліджень. У даній дисертаційній роботі застосовувалися методи обробки природної мови, засновані на правилах, словниках та існуючих лінгвістичних ресурсах, і ймовірнісних тематичних моделях, заснованих на комплексі методів машинного навчання.

Основні положення, що виносяться на захист:

а) запропоновано і реалізовано метод класифікації повідомлень на основі градієнтного бустінгу для української мови;

б) запропоновано і реалізовано метод класифікації текстових даних з потоку даних;

в) розроблено програмне забезпечення та проведено експериментальне дослідження, що обґрунтовують поліпшення якості запропонованих методів в порівнянні з існуючими алгоритмами.

Ступінь достовірності підтверджується коректністю розроблених методів і моделей, взаємозв'язком даних експериментів і наукових висновків, зроблених в роботі, результатами апробації алгоритмів і розробленого програмного прототипу систем. Результати експериментальних досліджень узгоджуються з результатами класифікацій відгуків.

Наукова новизна. Завдання визначення сентименту повідомлень з неперервного потоку даних мало досліджене в літературі. У даній роботі

запропоновані модифіковані методи обробки повідомлень на основі дерев з підтримкою української мови, що є його перевагою в порівнянні з іншими наведеними методами. Поліпшення якості розроблених методів в порівнянні з існуючими методами підтверджено експериментально за допомогою стандартних метрик якості систем аналізу текстів природною мовою. Експериментально показано, що розроблені методи застосовні до різних типів повідомлень з різних джерел, таких як групи телеграм канали, групи і т.д..

1 ОГЛЯД ІСНУЮЧИХ ПІДХОДІВ РОЗВ'ЯЗАННЯ ЗАДАЧІ

1.1 Передобробка потокових текстових даних

Процес предобробки вхідних даних є важливим та необхідним етапом в задачах обробки природної мови, на рівні з етапом самої класифікації чи розпізнання. Процес передобробки найчастіше використовується для виокремлення необхідної інформації з неструктурованих текстових даних. Найчастіше тексти зашумлені зайвими даними, такі як дати, числа, слова, які не несуть семантичного змісту - прийменники, артиклі, займенники.

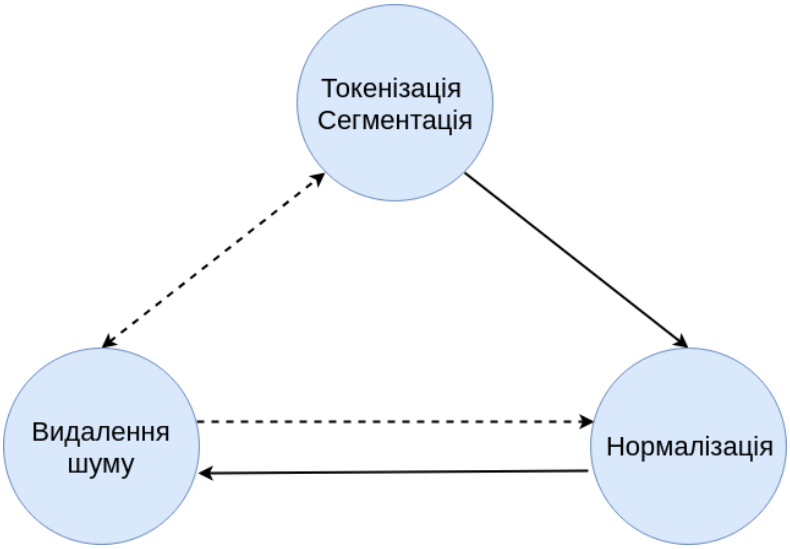


Рисунок 1.1 – Процесс предобробки тексту

Перший етап обробки називається токенізація (tokenization). Токенізація - це розбиття тексту на більш дрібні частини, токени. До токені відносяться як слова, так і знаки пунктуації.

Наступний етап – нормалізація (normalization). Нормалізацією називають процес трансформації тексту, в єдину канонічну форму, в якій він міг не бути спочатку. Процес нормалізації вимагає розуміння того, який текст

буде оброблятися далі, а який буде нормалізований. І в залежності від цього застосовують один або декілька з наведених нижче методів.

Зазвичай тексти містять різні граматичні форми одного і того ж слова, а також можуть зустрічатися однокореневі слова. *Лематизації* і *стемінг* мають на меті привести всі форми слова до однієї, нормальної словникової формі.

Стемінг - це грубий евристичний процес, який відрізає «зайве» від кореня слів, часто це призводить до втрати словотворчих суфіксів.

Слово	Стемінг
безпритульна	безпритул
повільне	повіл
ортогональний	ортогонал
цивільним	циві

Рисунок 1.2 – Приклад стемінгу

Лематизації - це більш тонкий процес, який використовує словник і морфологічний аналіз, щоб в результаті привести слово до його канонічної форми - леми.

Таблиця 1.1 - Приклад леми

Слово	Лема
розумієте	*розуміти*
повільне	повільний
цивільним	цивільне

Відмінність в тому, що стемер діє без знання контексту і, відповідно, не розуміє різницю між словами, які мають різний зміст в залежності від частини мови. Однак у стеммера є і свої переваги: його простіше впровадити і він

працюють швидше. Плюс, більш низька «точність» може не мати значення в деяких випадках.

Таблиця 1.2 – Порівняння леми та стемінгу

Слово	Лема	Стемінг
безпритульна	безпритульний	безпритул
особистість	особистість	особист
спогади	спогад	спогад

Стоп-слова - це слова, які викидаються з тексту до/після обробки тексту. Коли ми застосовуємо машинне навчання до текстів, такі слова можуть додати багато шуму або створити аномалії, тому необхідно позбавлятися від нерелевантних слів.

Під стоп-словами розуміють нецензурну лексику, вигуки, сполучники і т.д., які не несуть сенсу чи значеннях яких потрібно враховувати. Їх ще називають шумовими словами. При цьому потрібно розуміти, що не існує універсального списку стоп-слів, все залежить від конкретного випадку.

До типових стоп слів належать:

- цифри: 1, 2, 3, 4, 5, 6, 7, 8, 9, 0;
- знаки пунктуації розташовані окремо: . , = + /! "; :%? * ();
- окремо розташовані букви алфавіту;
- нецензурна мова.

Загалом, мета цих процедур – якнайбільше зменшення розмірності задачі без втрати емоційної складової.

1.2 Потоки текстових даних

Під потоком текстових повідомлень розуміється послідовність текстових повідомлень з певними для кожного повідомлення моментами часу. Під обробкою потоку текстових повідомлень розуміється комплексна задача оперативного аналізу сентименту вхідних повідомлень, визначення загального сентименту повідомлення, а також повідомлення про зміну сентименту всього потоку. Існує декілька робіт по способах обробки потоків даних. [1]

1.3 Мета та завдання сентимент-аналізу

Основною метою аналізу тональності тексту (сентимент-аналіз) є знаходження думок в тексті і виявлення їх властивостей. Наприклад, метою аналізу може бути автор, тобто особа, якій належить думка. Тобто, сентимент аналіз - це область комп'ютерної лінгвістики, яка займається вивченням думок і емоцій в текстових документах.

Текст на природній мові, крім інформації, може висловлювати емоційну оцінку того, про що повідомляється. Наприклад, таке речення містить негативну оцінку того, що відбувається:

Два роки назад наших спортсмені викрили у використанні допінгу.

А таке позитивне:

Автор видав нову збірку віршів, яка стала бестселером.

Виражена в тексті емоційна оцінка називається тональністю або сентиментом тексту. Людина оцінює світ відразу за багатьма показниками (хороший-поганий, сильний-слабкий, великий-маленький, щасливий-нещасливий, веселий-сумний, швидкий-повільний і т.п.), і показники ці мають

різне оціночне навантаження. Але для простоти можна вважати, що емоційна оцінка зводиться до шкали хороший-поганий або позитивний-негативний.

Об'єкт (entity, feature), щодо якого виражається емоційна оцінка, прийнято називати об'єктом тональності. Такий вид сентимент аналізу називається об'єктної тональністю.

Носієм (holder) вираженою в тексті емоційної оцінки також зазвичай є цілком певна особа, як правило це автор тексту. Однак якщо автор тексту посилається на чию-небудь думку, чи цитує висловлювання іншої людини, то носієм сентимент оцінки, або, як ще кажуть, суб'єктом тональності буде той, на чию думку посилаються.

Іноді класифікація відбувається в два етапи і на обох етапах є бінарної. На першому відокремлюються суб'єктивні повідомлення від об'єктивних. Об'єктивними в цьому випадку називаються якраз ті, які не несуть емоційного забарвлення і є нейтральними в варіанті з трьома класами. Другий етап ділить суб'єктивні тексти на позитивні і негативні. У випадку з Twitter, де майже всі повідомлення суб'єктивні, а критерії нейтральності можна сформулювати тільки в сенсі «не позитивно» і «не негативно», будемо для простоти розглядати поділ на два класи.

1.4 Соціальна мережа Twitter

Твіт - це рядок, що складається з не більше ніж 280 символів. Він може містити спеціальні слова, що починаються з певних знаків: відразу після «@» пишеться ім'я користувача, з яким пов'язано повідомлення або до якої воно звернено, а після «#» знаходиться так званий хештег - слово, яке явно вказує на зв'язок твіта з об'єктом, який цим словом позначається.

Всі твіти створюються користувачами, тому можуть містити помилки, скорочення, особливу пунктуацію і інші способи вираження думки в короткому тексті. У кожного повідомлення в Twitter є час, коли воно опубліковано, і автор. Якщо один твіт є відповіддю на інший, то у першого є посилання на другий, тобто на «батьківський». Ретвіти також містять дані про початкове розміщення. Поставлена задача вирішується обчислювально, за допомогою технік машинного навчання. Перша згадка завдання аналізу тональності відноситься до 2002 року. Тоді були розглянуті стандартні рішення методом навчання без вчителя і методом навчання з учителем. В обох статтях досліджувалися відгуки на спеціалізованому ресурсі: метою була задача визначення чи рекомендує користувач, який залишив відгук, те, про що він написав.

1.5 Способи класифікації текстів за тональністю

Сьогоднішні автоматизовані системи зазвичай використовують одновимірний простір для визначення емоційності тексту: позитивний чи негативний контекст (іншими словами, добрий чи поганий). Але є також успішні випадки використання та n-мірного простору. Основне завдання аналізу настроїв - класифікувати полярність розглянутого документа, тобто визначити, чи є позитивні думки, висловлені в документі, чи негативними чи нейтральними. Більш широко, наприклад, n-мірна класифікація тонів може виражатися такими емоційними станами: "злий", "сумний" і "щасливий".

1.5.1 Класифікація за бінарною шкалою

У цьому випадку полярність документа можна визначити у двійковій формі. Для визначення полярності документа є два класи оцінок: позитивний або негативний. Одним з недоліків такого підходу є те, що емоційний компонент документа не завжди може бути чітко визначений, тобто документ може містити як позитивні, так і негативні ознаки. Рання робота в цій галузі включала обов'язки Терні[5] та Панга[6], які застосовують різні методи розпізнавання полярності оглядів товару і відгуків про фільми відповідно. Це приклад роботи на рівні документа.

1.5.2 Класифікація за багатополосною шкалою

Полярність документа також може бути класифікована в багатополосною шкалою, запропонований Пангом [6] та Снайдером [7]. Вони розширили фокус оглядів рейтингів фільмів з "позитивних чи негативних" в сторону прогнозування рейтингу n-бальною шкалою. У той же час Снайдер провів аналіз оглядів ресторанів, надавши рейтинги різних ознак, таких як їжа та атмосфера (5-бальна шкала).

1.5.3 Системи шкалювання

Іншим способом визначення інтонації є використання системи шкалювання, за якою слова, які зазвичай асоціюються з негативними, нейтральними чи позитивними інтонаціями, будуть відповідати числам від -10 до 10 (від найвід'ємнішої до найбільш позитивної). Спочатку засоби для обробки природних мов та алгоритми використовуються для вивчення фрагментів неструктурованого тексту, а потім об'єкти та терміни, витягнуті з тексту, аналізуються для розуміння значення слів.

1.5.4 Суб'єктивність/об'єктивність

Інший напрям дослідження - виявлення суб'єктивності / об'єктивності. Це завдання, як правило, визначається для віднесення зазначеного тексту до однієї з двох категорій: суб'єктивній чи об'єктивній. Іноді ця проблема може бути складнішою, ніж класифікація полярності: суб'єктивність слів і фраз може залежати від їх контексту, тоді як об'єктивні документи можуть містити суб'єктивні пропозиції (наприклад, стрічки новин, які цитують думки інших). Більше того, результати більшою мірою залежать від суб'єктивних визначень, які використовуються в рамках текстових анотаціях. Панг зробив усе можливе, щоб показати, що видалення об'єктивних пропозицій з документа перед класифікацією полярності допомагає підвищити точність результатів.

Модель більш докладного аналізу називається аналізом на основі функції/аспекту. Модель посилається на визначення настроїв чи думок, виражених різними аспектами або функціями сутностей, наприклад, у смартфону, цифрової камери. Атрибути / аспекти - це найважливіші атрибути або елементи, перевірені на тональність, наприклад, екран мобільного телефону або найкращий тип камери. Більш докладні дискусії з цього приводу

можуть бути знайдені в довіднику по NLP, в розділі «Аналіз тональності і суб'єктивності».

1.6 Існуючі методи класифікації тональності

1.6.1 Метод Опорних Векторів

Метод Опорних Векторів(SVM) працює за принципом машинного навчання з учителем. Він використовується для бінарної класифікації. SVN працює за принципом поділу простору на підпростори, які відповідають класам. Точки побудовані на поверхні називаються опорними векторами. Далі робота йде вже з цими векторами і простором, в якому вони розташовуються. Він може замінювати нейронні мережі, але процес навчання у нього дуже повільний.

Знаходження параметрів SVM відповідає опуклій оптимізації. Завдання класифікації, як правило, включає в себе поділ даних на навчальні та тестові набори. Кожен екземпляр в навчальному наборі містить «одне цільове значення» і кілька атрибутів. Основною метою SVN є вироблення моделі, на основі навчальних даних, яка визначає цільове значення тексту, та побудова оптимальної розмежувальної гіперплощини. Основним завданням SVN є знаходження лінійної моделі наступного вигляду:

$$y(x) = w^y x + b \text{ . (1.1)}$$

Де x, вхідний вектор, w і b параметри, котрі корегуються для певних моделей, що оцінюються емпіричним шляхом.

Для простої лінійної класифікації завдання полягає в тому, щоб звести до мінімуму функцію помилок, що визначається за рівнянням:

$$C\sum_{n=1}^{N}\xi_n + \frac{1}{2}||\omega||^2 \rightarrow min\ . (1.2)$$

Де С — константа;

ω — вектор коефіцієнтів;

ξ — параметр для обробки неподільних даних (входів);

n — номер процедури навчання.

Даний метод опорних векторів широко використовують в різноманітних задачах машинного навчання, в яких він показує гарні результати.

SVM відрізняється від інших гіперплощинних методів класифікації тим, що він дозволяє обирати оптимальне розташування гіперплощини [4]. Гіперплощина обирається таким чином, щоб бути розташованою на максимальній відстані від елементів кожного з класів, тобто посередині деякої зони, що відділяє між собою ці елементи (на рис. 1.3 граничні елементи заретушовані).

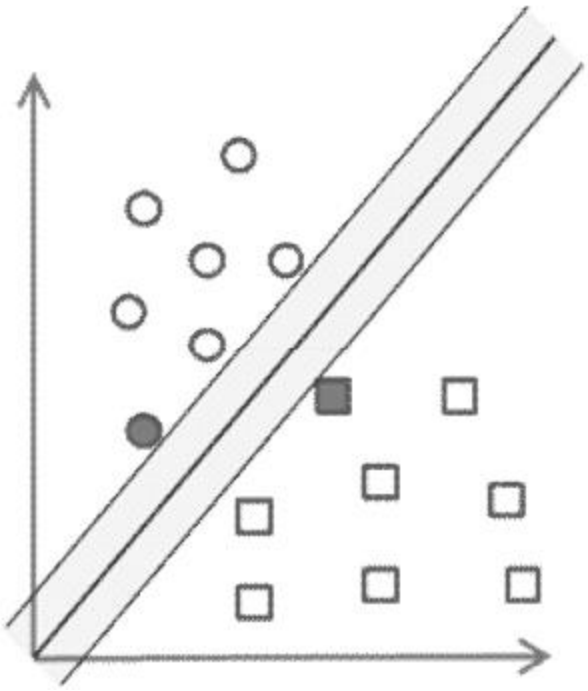


Рисунок 1.3 – Гіперплощина

Якщо тренувальні дані є лінійно роздільними, то ми можемо обрати дві паралельні гіперплощини, які розділяють два класи даних таким чином, що відстань між ними є якомога більшою. Область, обмежена цими двома гіперплощинами, називається «розділенням» (англ. "margin"), а максимально роздільна гіперплощина є гіперплощиною, яка лежить посередині між цими двома[3]. На рисунку 1.4 зображена максимальна розділова гіперплощина та межі, натренованої зразками з двох класів. Зразки на межах називаються опорними векторами.

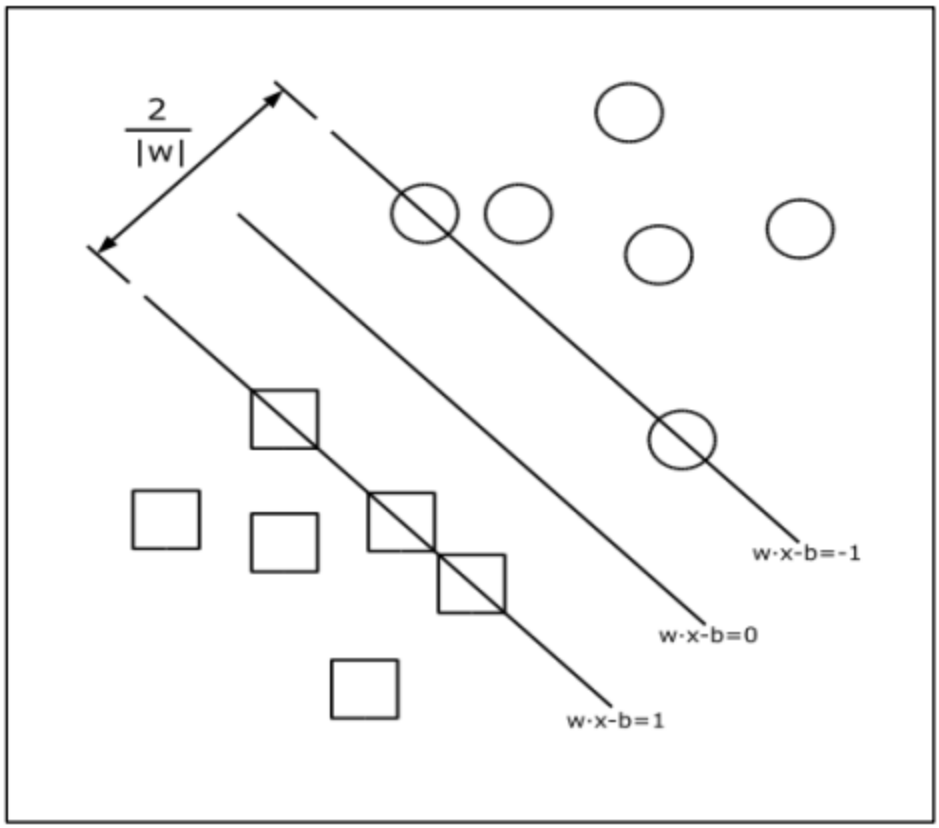


Рисунок 1.4 - Максимальна розділова гіперплощина

Даний метод являється одним з найпопулярніших в задачах бінарної класифікації та показує гарні результати.

1.6.2 Методи, засновані на правилах та словниках

Полягає в обчисленні настрою тексту в залежності від полярності слів або фраз в тексті. Методика полягає у наступному: після попередньої обробки тексту, відбувається перевірка маркера кожного слова на його полярність в лексиконі. Якщо слово не знайдено в лексиконі, тоді полярність слова вважається нульовою. Після призначення балів полярності W всім словам, остаточна оцінка S настрою тесту розраховується діленням суми балів слів, які задають настрій тексту на кількість таких слів.

$$S = \frac{1}{m}\sum_{i=1}^{m} {W_i}^1 \text{ . (1.3)}$$

Де W_i — бал полярності і-го слова; m – кількість слів, які задають настрій тексту.

У середньому балу дозволяє отримати числове значення балу настрою у діапазоні від -1 до 1, де 1 означає сильний позитивний настрій, -1 означає сильний негативний і 0 означає що текст є нейтральним. Якість класифікації багато в чому залежить від якості словника.

Великий електронний словник української мови (**ВЕСУМ**). Словник бере початок з проекту ispell-uk, що його в 90-х роках створила група ентузіастів для перевірки орфографії української мови у відкритій ОС Linux.

Підчас творення словника використовувалися «Граматичний словник української літературної мови. Словозміна» та «Словники України».

Словник налічує понад 285 тисяч лем та постійно оновлюється, також розміщений на Github[10], під відкритою ліцензією, що дозволяє приєднатися до його створення.

Головним недоліком методів на основі словників та правил є складність процесу формування словників. Для отримання методу, який здатний класифікувати документ з достатньою точністю, терміни у словнику повинні мати досить велику вагу, яка буде достатньою для предметної області документа. Наприклад, термін "величезний" у відношенні до об'єму пам'яті жорсткого диска є позитивною характеристикою і контекстом, але негативним відносно габаритів смартфону. Саме з цієї причини, даний метод вимагає великих зусиль, оскільки для задовільної роботи системи треба створити немалу кількість правил. Але існує ряд алгоритмів, які покликанні,

автоматизувати процес складання словників для конкретної предметної області.

1.6.3 Дерева прийняття рішень

Дерева рішень можуть бути адаптовані до практично будь-якого типу даних, тому цей спосіб широко використовується в алгоритмах машинного навчання. При контрольованому машинному навчанні використовуються алгоритм, який ділить виготовлені дані на більш дрібні частини, з метою визначення моделі, яка може бути використана для класифікації. Дані потім представляються у вигляд логічних структур, подібних до деревовидної, які можуть бути легко зрозумілі без будь-яких статистичних знань. Алгоритм особливо добре підходить для випадків, коли може бути знайдено багато ієрархічних категоріальних відмінностей. Вони побудовані з використанням евристичних алгоритмів, які називають рекурсивним розбиття. Це, як правило, відомо, як підхід «розділяй і володарюй», оскільки він використовує значення функцій для поділу даних на менші підмножини подібних класів. Структура дерева рішень складається з кореневого вузла, який представляє собою весь набір даних, рішень вузлів, які виконують обчислення і листових вузлів, які здійснюють класифікацію [11].

Для того, щоб класифікувати невідомий екземпляр, дані передаються через дерево. Кожному вузлів рішення певної функції, отриманої з вхідних даних, порівнюються з константою, яку було визначено на етапі підготовки. Обчислення, яке відбувається в кожному вузлів рішення, зазвичай, порівнює обрану функцію з цією, заздалегідь заданою, константою, тоді рішення буде гуртуватися на функції, створюючи два способи поділу на дереві. Дані будуть, в кінцевому підсумку, проходити через ці вузли рішення до тих пір, доки не

досягнуть листового вузла, який представляє собою визначений клас. Є багато різних реалізацій і варіантів алгоритму дерев рішень, такі як, Random Forest та метод J48, який є реалізацією Java алгоритму C4.5. Для виявлення спаму, пропаганди, інформаційно-психологічних впливів дерева прийняття рішень на практиці застосувати неможливо, оскільки вони не підтримують інкрементного навчання [11]. Можна взяти великий об'єм даних та побудувати для нього дерево рішень, але врахувати нові повідомлення даний метод не зможе, оскільки для нових даних його доведеться кожен раз навчати заново.

1.6.4 Наївний класифікатор Баєса

В якості найпростішого методу для класифікації тональності тексту. А також фільтрації спаму, використовується наївний класифікатор Баєса. У даному класифікаторі використовується теорема Баєса для визначення ймовірності приналежності елемента вибірки до одного з класів при припущенні незалежності ознак. Для підвищення якості класифікації застосовується метод максимальної ентропії [11]. Класифікатор максимальної ентропії є класифікатором ймовірності, який належить до класу експоненціальна моделі. На відміну від наївного класифікатора Баєса, він не припускає, що ознаки умовно незалежні одна від одної. Цей класифікатор засновано на принципі максимальної ентропії усіх моделей, які відповідають даними навчання з найбільш рівномірним розподілом. Класифікатор максимальної ентропії може бути використаний для вирішення великої кількості різноманітних завдань класифікації тексту, таких як виявлення спаму, сленгу, тематичної класифікації.

1.6.5 Метод k-найближчих сусідів

kNN - це один з найпростіших алгоритмів класифікації, також іноді використовується в задачах регресії. Завдяки своїй простоті, він є хорошим прикладом, з якого можна почати знайомство з областю Machine Learning.

Для класифікації кожного з об'єктів тестової вибірки необхідно послідовно виконати наступні операції:

- обчислити відстань до кожного з об'єктів навчальної вибірки;
- відібрати k об'єктів навчальної вибірки, відстань до яких мінімальна.

Клас об'єкту класифікації - це клас, який найчастіше трапляється серед k найближчих сусідів.

kNN - один з найпростіших алгоритмів класифікації, тому на реальних завданнях він часто виявляється неефективним. Крім точності класифікації, проблемою цього класифікатора є швидкість класифікації: якщо в навчальній вибірці N об'єктів, в тестовій виборі M об'єктів, а розмірність простору - K, то кількість операцій для класифікації тестової вибірки може бути оцінений як O(K*M*N).

Переваги та недоліки дослідних методів автоматичного визначення тональності тексту контенту у віртуальних соціальних мережах представлені у таблиці 1.3.

Таблиця 1.3 – Переваги та недоліки кожного з методів

Назва методу	Необхідність застосування словників	Необхідність попередньої лінгвістичної обробки тексту	Можливість застосування до різних типів даних
Лексемний метод	+	+	Тільки до текстів
Метод опорних векторів	-	-	До різних типів даних
Дерева прийняття рішень	-	+	До різних типів даних
Наївний класифікатор Баєса	+	-	Тільки до текстів
kNN	-	-	До різних типів даних

Висновки за розділом

Аналіз тональності тексту - клас методики контент-аналізу в текстовій лінгвістиці, що дозволя визначати емоційне забарвлення тексту на основі його складових, слів.

У розділі детально описано процес передобробки текстових даних, до подачі їх на вхід класифікатору (лематизація, токенізація, стемінг) визначення емоційного забарвлення. Описано існуючі підходи для сентимент аналізу, проаналізовано можливі методи та алгоритми аналізу тексту, а також наведено детальну характеристику кожного з методів та його можливостей. Проведено порівняння всіх проаналізованих алгоритмів для сентимент аналізу.

2 ОБРОБКА ТЕКСТОВИХ ПОТОКІВ ДАНИХ

У цьому розділі проведена оцінка можливих інструментів для обробки текстових потоків даних, для кожного інструменту чи засобу наведена характеристика та опис, в кінці розділу порівняно їх за основними критеріями оцінки а також вибрано найбільш підходящий під задачу.

2.1 Apache Storm

Apache Storm - це обчислювальна система в режимі реального часу з відкритим кодом для обробки потоків даних. Подібно до того, що робить Hadoop для батч-обробки, Apache Storm забезпечує надійний обмін безграничними потоками даних, тобто він не поділяє потоки на частинки як це робить приміром Hadoop [12].

- Apache Storm здатний обробити понад мільйон завдань на вузлі за частку секунди.
- Він інтегрований з Hadoop для використання більшої пропускної здатності.
- Це легко здійснити і може бути інтегровано з будь-якою мовою програмування.

Apache Storm дозволяє виконувати масштабування горизонтально, що дозволяє продовжувати обчислення паралельно з однаковою швидкістю при збільшенні навантаження. Як згадувалося раніше, важливо відзначити, що він створив еталон обробки 1 мільйона повідомлень розміром 100 байт на одному вузлі, що робить його однією з найшвидших технологічних платформ. Володіючи такими характеристиками швидкості і масштабованості, ця технологія затьмарює інші існуючі, коли мова заходить про обробку великих обсягів даних з безпрецедентною швидкістю.

2.2 Apache Hadoop

Apache Hadoop являє собою набір утиліт з відкритим вихідним кодом, які полегшують використання мережі з багатьох комп'ютерів для вирішення проблем, пов'язаних з великими обсягами даних і обчислень [13].

Коли ми говоримо про Hadoop, то в першу чергу маємо на увазі його файлову систему – HDFS. Найпростіший спосіб думати про HDFS - це уявити звичайну файлову систему, тільки більше. Звичайна ФС, за великим рахунком, складається з таблиці, файлових дескрипторів і області даних. У HDFS замість таблиці використовується спеціальний сервер - NameNode, а дані розкидані по серверам даних (DataNode).

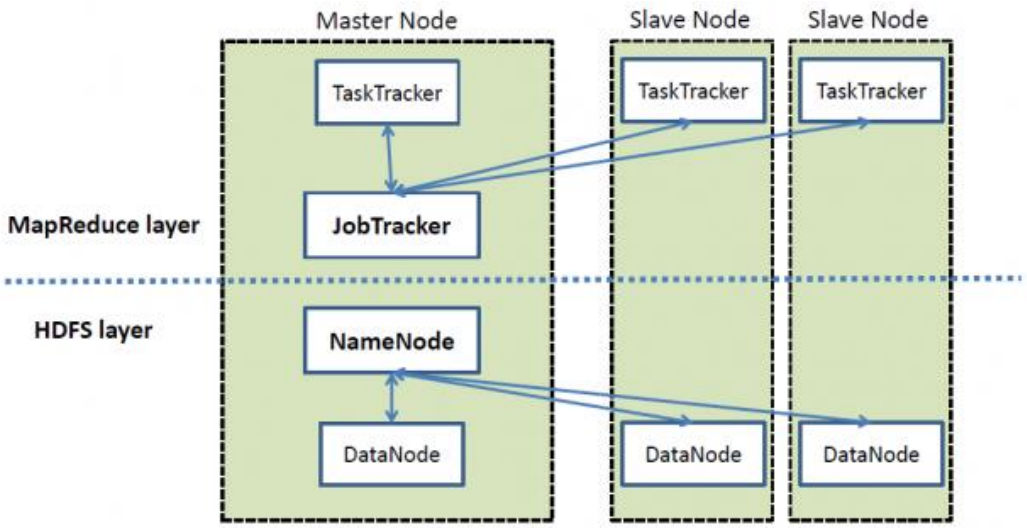


Рисунок 2.1 - Архітектура Hadoop вищого рівня

Hadoop streaming - це універсальний API, який використовується для роботи з потоковими даними. І картограф, і редуктор отримують свої вхідні дані у стандартному форматі. Вхідні дані беруться з stdin і висновок в stdout.

Hadoop-це програма, яка використовується для обробки і зберігання великих даних. Розробка Hadoop - це завдання обчислення великих даних з використанням різних мов програмування, таких як Java, Scala і інших.

2.3 Splunk

Splunk на відмінну від усіх інших представлених вище засобів не належить до Apache сім'ї.

Надбудова Splunk для веб-сервера Apache дозволяє адміністратору програмного забезпечення Splunk збирати і аналізувати дані з веб-сервера Apache за допомогою моніторингу файлів. Це доповнення надає вхідні дані і CIM-сумісні знання для використання з іншими додатками Splunk.

Splunk Stream - це спеціально створене рішення для збору даних і аналітики логів від Splunk. Пасивне захоплення пакетів, динамічне виявлення додатків, аналіз протоколу і відправка метаданих назад в індексатор для більш ніж 30 протоколів [14].

Splunk добре використовувати для аналізу логів програм, для обробки текстових потоків даних його можна використовувати але дуже важко, тому що він не був створено.

2.4 Amazon Kinesis

Amazon Kinesis також не відносить до сімейства Apache.

Amazon Kinesis спрощує збір, обробку та аналіз потокових даних в режимі реального часу, що дозволяє своєчасно отримувати інформацію і швидко реагувати на нову інформацію.

Amazon Kinesis може збирати і обробляти сотні гігабайт даних в секунду з сотень тисяч джерел, що дозволяє легко створювати додатки, що обробляють

інформацію в режимі реального часу, з таких джерел, як потоки кліків на веб-сайті, маркетинг і фінансова інформація, виробничі прилади та соціальні мережі, а також операційні журнали і дані вимірювань [15].

Головним недоліком Kinesis для виконання поставленої задачі, а саме обробки текстових потоків даних є важкість в виконані завдання. Ще одним недоліком Kinesis є його ціна в порівнянні з іншими продуктами, що і не дивно, враховуючи те що ми працюємо з AWS. Потрібно руками оптимізувати та налаштовувати його роботу, що зменшити ціну за використання, так наприклад можна агрегувати повідомлення розміром менше 25кб в одне. Приклад архітектури Kinesis наведений нижче.

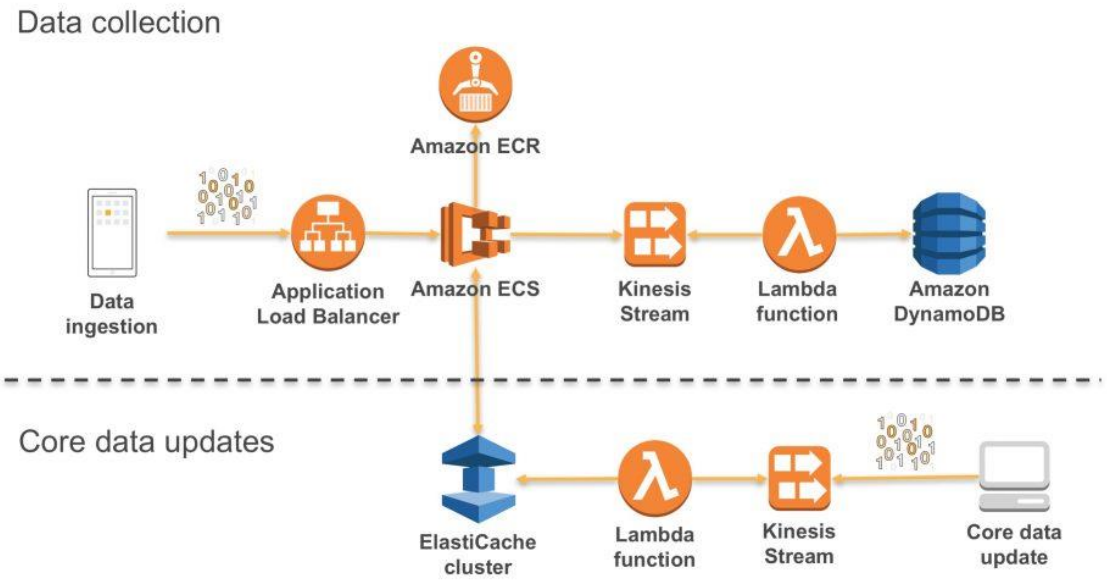


Рисунок 2.2 - Приклад системи на базі Kinesis

Головною перевагою, та водночас недоліком Kinesis є його належність до сервісів AWS, що надає змогу легко інтегруватись з іншими

амазоновськими сервісами, і стає важко інтегруватися з сервісами не amazon, наприклад ви не зможете як в старі добрі часи підняти в себе в підвалі на старому комп’ютері кластер ElasticSearch, вам обов’язково потрібно використовувати реалізацію ES від амазону.

2.5 Apache Spark

Apache Spark-це платформа розподілених кластерних обчислень з відкритим вихідним кодом. Spark - це механізм обробки даних, розроблений для забезпечення більш швидкої і простий у використанні аналітики, ніж Hadoop MapReduce[16].

Головні переваги Apache Spark наведені нижче.

- Обробка в пам'яті: обробка в пам'яті виконується швидше в порівнянні з Hadoop, так як немає часу, що витрачається на переміщення даних / процесів на диск і з нього. Spark в 100 разів швидше, ніж MapReduce, так як все робиться тут в пам'яті.
- Передача потокового обробка: Apache Spark підтримує потокову обробку, яка включає безперервний введення і виведення даних. Передача потокового обробка також називається обробкою в реальному часі.
- Менша затримка: Apache Spark щодо швидше, ніж Hadoop, оскільки він кеширует більшу частину вхідних даних в пам'яті за допомогою еластичного розподіленого набору даних (RDD). RDD управляє розподіленою обробкою даних і їх перетворенням. Саме тут Spark виконує більшість операцій, таких як перетворення і управління даними. Кожен набір даних в RDD розбивається на логічні частини, які потім можуть бути обчислені на різних вузлах кластера.
- Лінива оцінка: Apache Spark починає оцінювати тільки тоді, коли це абсолютно необхідно. Це відіграє важливу роль у сприянні його швидкості.

- Менше рядків коду: хоча Spark написаний як на Scala, так і на Java, реалізація знаходиться в Scala, тому кількість рядків в Spark щодо менше в порівнянні з Hadoop.

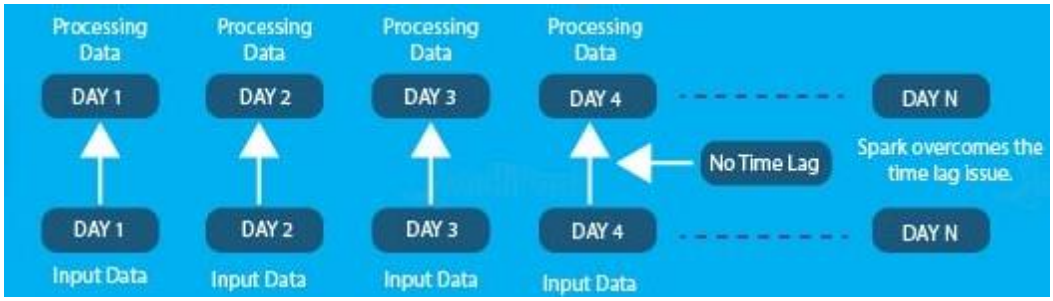


Рисунок 2.3 – Обробка в режимі реального часу на Spark

Також перевагою Spark є широкий вибір його компонентів, таких як:

- Spark Core – ядро спарку;
- Spark SQL – для виконання запитів до даних (потоків, графів...), на SQL подібному синтаксисі;
- Spark Streaming – для розподіленої обробки потоків даних, підходить для вирішення поставленої задачі;
- Spark GraphX для розподіленої обробки графових структур;
- Spakr MLib – бібліотека для машинного навчання, з широким вибором вже існуючих та реалізованих алгоритмів.

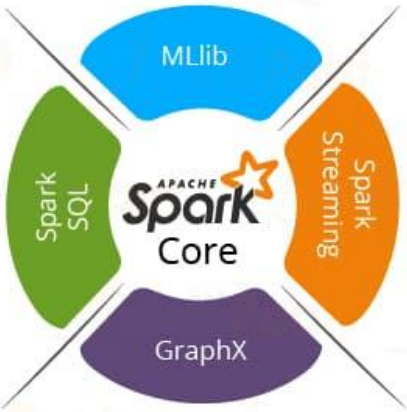


Рисунок 2.4 - Компоненти Spark

Незважаючи на всі переваги Apache Spark, існують і певні недоліки в його використанні, а саме відсутність власної файлової системи, тому доводиться інтегрововуватись з чимось на кшталт HDFS або ж хмарними сховищами (S3).

У разі Apache Spark вам потрібно оптимізувати код вручну, так як він не має ніякого автоматичного процесу оптимізації коду. Це обернеться недоліком, коли всі інші технології та платформи будуть рухатися в бік автоматизації.

2.6 Apache Hive

Apache Hive - система управління базами даних на основі платформи Hadoop. Дозволяє виконувати запити, агрегувати і аналізувати дані, що зберігаються в Hadoop. Традиційно додавання нових даних в Hive вимагає збору великої обсягу даних в HDFS, а потім періодичного додавання нового розділу. Це, по суті, "пакетна вставка" [17]. Після фіксації даних вони відразу стають видимими для всіх запитів Hive, ініційованих згодом.

2.7 Apache Flink

Apache Flink - це платформа обробки потоків з відкритим вихідним кодом для розподілених додатків з високим ступенем відмово стійкості і толерантністю до падінь.

І хоча Flink побудований на базі потокового движка, аналіз потоків не є для нього головним.

Замість цього Flink націлюється на додатки з запам'ятовуванням станів, оскільки на відміну від інших потокових движків з відкритим вихідним кодом він підтримує фіксацію часу подій. Це означає, що програми, що працюють з каналами передачі даних, можуть робити відкат і повтор.

Кластер flink можна запустити в yarn, mesos або за допомогою окремих (вбудованих в пакет flink) task- і job-manager's.

Тобто головною перевагою Flint можна назвати його унікальність в цьому плані, він єдиний підтримує стан потоку, що допомагає йому рухатись в зворотному напрямку потоку[18].

Недоліком ж його є дуже довгий час після виходу нових версій та й самий проект ще молодий і важко підходить до вирішення задач промислового рівня.

Висновки за розділом

В розділі було порівняно аналогічні програмні засоби системи для потокової обробки текстових потоків даних, наведено їх основні недоліки та плюси.

Серед запропонованих варіантів було обрано програмний комплекс Apache Spark, оскільки має всі необхідні можливості для виконання поставленої задачі, а саме обробку в оперативній пам’яті, гнучкість в масштабуванні та налаштуванні системи. З «коробки» є вся необхідна інфраструктура для виконання аналізу а також можливість інтеграції з іншими системами які будуть використані в процесі дослідження, Elastic Search та Kafka. Переваги, що надають його конкуренти, для дослідження не релевантні. Тому Apache Spark найбільш підходящий варіант для виконання дослідження.

3 МАТЕМАТИЧНІ МОДЕЛІ

3.1 Постановка задачі обробки потоків даних

Ми називаємо тестовим потоком даних

$$S = \{(d_0, t_o), (d_1, t_1), (\dots)\}. \quad (3.1)$$

нескінченний потік даних які надходять з одного або декількох джерел, де пара (m_j,t_j) визначає, що отримане повідомлення m_j отримано в точці часу t_j.

Часовим вікном називається W_i, часовий інтервал з фіксованим розміром δ, який починається у точці t_i.

Запропонований підхід для аналізу текстових потоків даних [1], полягає в запуску, нового незалежного циклу початкового завантаження *(bootstrapping cycle)* B_i для кожного часового вікна W_i . Кожний B_i відповідає за обробку тестового фрагменту, повідомлення M_i, що є частиною потоку S, який складається з повідомлень отриманих в W_i. Тобто можна сказати що,

$$M = \{m_j \mid j \in [i, i + \delta]\}. \quad (3.2)$$

Для кожного фрагменту повідомлення в повідомленні, $m_j \in M_i$, існує відповідний тестовий контент, наприклад повідомлення в чаті групи, каналі, твіт, пост на стіні, який зберігається в Data Stream Repository, з часом отримання, та ключовими словами K_{mj}, які були отримані в процесі перед обробки тексту, нормалізації та приведення до нижнього регістру літер.

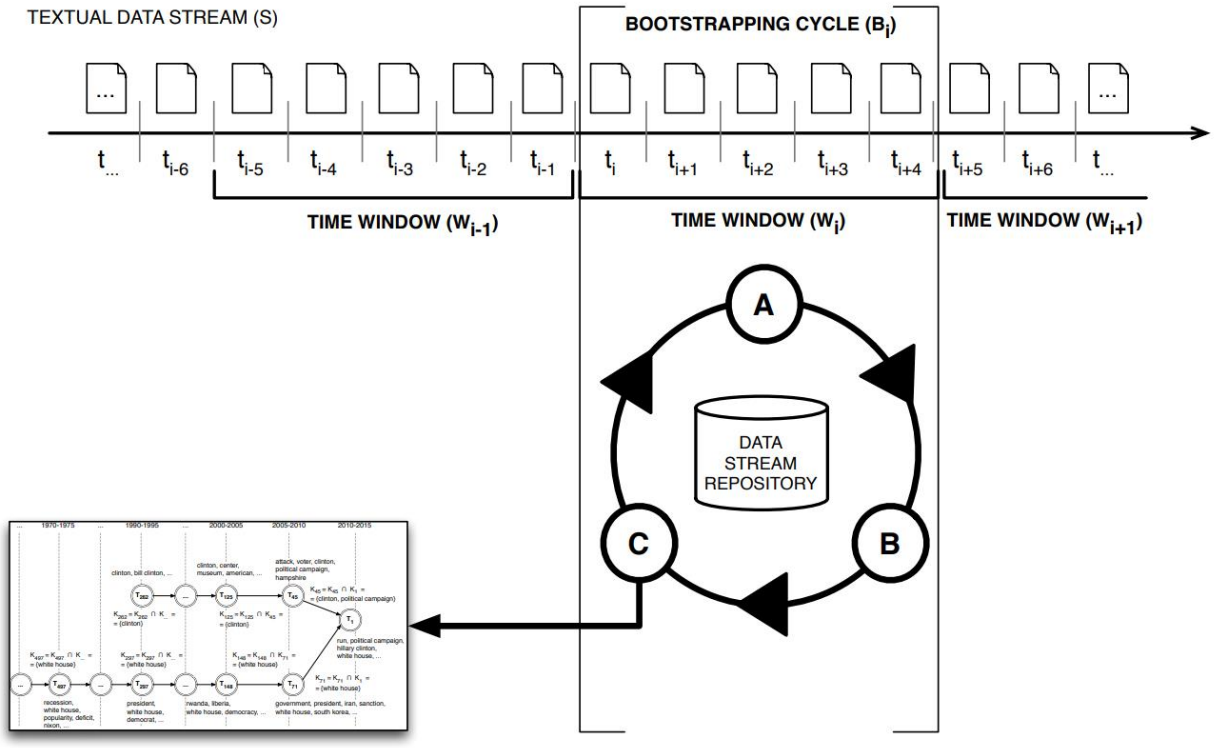


Рисунок 3.1 - Завантаження тестового потоку даних

3.2 Постановка задачі класифікації тестових повідомлень

Завдання класифікації текстів можна сформулювати як приближене завдання невідомої функції Ф: D×C→{-1,1} (який клас повинен бути в документа) через функцію K: D ×C→{-1,1}}, яка називається класифікатором, де $C = \{c_1, c_2, \dots, c_{|C|}\}$, сет можливих категорій, я нашому випадку він визначений явно (від дуже злої до дуже доброї) і $D = \{d_1, d_2, \dots, d_{|D|}\}$ – сет вхідних повідомлень.

$$\Phi(d_j, c_j) = \begin{cases} 1, d_j \in c_j \\ 0, d_j \notin c_j \end{cases}. \quad (3.3)$$

Документ (повідомлення) d_j називається поганим якщо $\Phi(d_j,c_j) = -1$.

Документ (повідомлення) d_j називається добрим якщо $\Phi(d_j,c_j) = 1$.

Для кожного повідомлення може бути визначена лише одна оцінка сентименту.

3.3 Модель латентного розподілу Діріхле

Модель латентного розподілу Діріхле (LDA) - це загальна модель, в якій кожен документ представлений у вигляді поєднання різних тем. Зразок Гіббса використовується для визначення параметрів LDA для набору документів.

Основний недолік латентного розподілу Дирихле - відсутність мовного виправдання, хоча є розширення, які знімають деякі обмеження та підвищують продуктивність для конкретних завдань. Латентний розміщення Діріхле є базовою ймовірнісної тематичної моделлю і через велику кількість додатків і узагальнень є найпоширенішою ймовірнісної тематичної моделлюОсновні імовірнісні тематичні моделі дозволяють визначити приховані теми для документа на основі шаблону документа у вигляді набору слів. Вони також пропонують приховані зв'язки між різними об'єктами, які можуть з’являтися у структурі вживання слів. Семантичну конвергенцію різних об'єктів можна оцінити, порівнюючи тематичні вектори.

На рисунку 3.2 зображено графічна ймовірнісна модель латентного розподілу Дирихле, де w - слово, t - тема (прихована змінна), D - колекція документів, N - довжина документу в словах, K - кількість тем в колекції, Θ - розподіл тем в повідомлені, Ф - розподіл слів в темі, α- апріорний розподіл Дирихле на параметриΘ, β - апріорний розподіл Дирихле на параметри Ф [17].

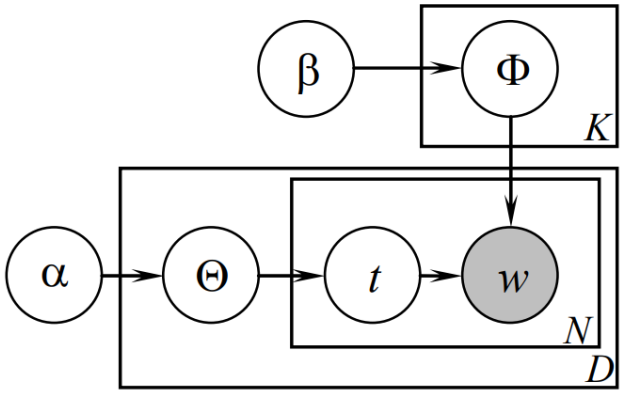


Рисунок 3.2 - Графічна ймовірнісна модель латентного розподілу Дирихле

Для ідентифікації параметрів моделі LDA по колекції документів було застосовано семплірування за Гіббсом - алгоритм для генерації вибірки спільного множино-випадкового розподілу випадковиї значень. Він використовується для оцінки спільного розподілу та для обчислення інтегралів за методом Монте-Карло. Потрібно визначити кількість тем в наборі документів, алгоритм семплірування за Гіббсом має наступні кроки:

а) для кожного слова з кожного документа привласнити випадково одну тему (t) з K можливих;

б) для кожного слова з кожного документа обчислити:

1) p (t | d) - долю слів у документі d, які присвоюються темі t;

2) p (w | t) - долю слів у всіх документах, присвоєних темі t;

3) присвоїти слову w новую тему з вірогідністю p (t | d) × p (в)

повторити другий крок кілька разів (кількість ітерацій).

3.4 Формування набору даних

Однією з головних труднощів у вирішенні завдань класифікації на основі методів вивчення конкретних випадків є пошук необхідних даних, за допомогою яких ми зможемо навчити нашу систему. Часто їх потрібно тривати довго. У нашому випадку є достатньо даних для онлайн-навчання. Просто перейдіть на першу сторінку пошуку форумів та соціальних мереж і швидко знайдіть перші приклади. Однак вам слід визначити їх та виконати роботу, необхідну для підготовки та позначення існуючих коментарів як належних до категорій зловживань.

Пошук готового набору даних увінчався успіхом, на сайті Github[9] було знайдено відкритий набору даних україномовних коментарів, постів та повідомлень з різних соціальних мереж. Набір даних містить близько 4 тисяч повідомлень, як з позитивними так і негативними емоціями, а також їх оцінку, що дозволить доволі швидко та правильно навчити класифікатор.

Кожний приклад складається з реального тексту реальної людини, та позначки емоційної складової, (0 – злий, 1 – добрий).

Для того що б протестувати правильність визначення емоційної складової повідомлень, потрібно з основного набору даних виділити тестові дані, тобто дані на яких буде перевірятись правильність навченої моделі. Для цього весь набору даних було розділено у співвідношенні 70/30, де 70 дані для навчання, а 30 – валідаційні, розподіл даних виконано у випадковому порядку, що дозволить уникнути колізій при повторному перезапуску чи перенавчанні класифікатора.

На рисунку нижче зображено приклад повідомлень, на яких і буде класифікатор навчатись.

```
{
    "isPositive": false,
    "message": "Сьогодні в школі на першому уроці мені єдиній розказала одна однокласниця що вчора її вистижили і напали в неї подряпини на ши
}
{
    "isPositive": false,
    "message": "Якщо бабця не очолює траст це поганий траст хіба не зрозуміло Здається іноді що коли вана зникне життя в країні автоматично на
}
{
    "isPositive": false,
    "message": "Ні ні ні саме воно я на цій темі не одній ваті розтин робив до самої сраки Колчак там дуже фігню робив так пишуть що це через
}
{
    "isPositive": false,
    "message": "там все просто цей цар поганий тре нового царя"
}
{
    "isPositive": false,
    "message": "Ранкова розмова з матір'ю про те що наш дід по маминій лінії був поганий Він був затриманий поліцією багато разів через те що
}
{
    "isPositive": true,
    "message": "читав роки півтори назад мені сподобалось(хоча я і Братів Карамазових три рази читав так що я поганий приклад)"
}
{
    "isPositive": false,
    "message": "Через поганий авторитет Онищенка компромат на Порошенка не став гучним журналіст"
}
{
    "isPositive": false,
    "message": "Чим договір з лікарем поганий"
}
{
    "isPositive": false,
    "message": "Кримінальна відповідальність за поганий ремонт доріг КМДА пояснила що чекає на підрядчиків бракоробів"
}
{
    "isPositive": false,
    "message": "Кримінальна відповідальність за поганий ремонт доріг КМДА пояснила що чекає на підрядчиків бракоробів"
}
{
    "isPositive": true,
    "message": "Подивився нарешті цей фільм настільки поганий що геніальний рекомендую"
}
{
    "isPositive": false
```


Рисунок 3.3 - Приклад повідомлень для навчання класифікатора

3.5 Обґрунтування вибору класифікатора

При вивченні сучасних моделей машинного навчання було встановлено, що найбільш популярними є: Gradient Boosting Tree, SVM (векторний метод) та Logistic Regression (логістична регресія) для класифікації тексту.

Щоб визначити, який метод машинного навчання слід використовувати як основу для вирішення проблеми класифікації тексту за емоційними ознаками, маючи в ньому зображення, необхідно застосувати кожен із трьох методів та оцінити основні показники базових моделей. Оскільки на цьому

етапі вам потрібно лише визначити найкращий метод, навчальний вхідний текст буде підданий базовій обробці.

Перехресне валідацію було обрано як міру оцінки якості, що дозволяє оцінити точність класифікації даних, які не були представлені перед робочим процесом. Це дає можливість отримати більш реалістичні показники якості класифікатора. Точність класифікації також вимірювали для порівняння.

Порівнюючи показники точності на навчальному наборі та точності виміряної за допомогою перехресної валідації стає очевидним, що в кожному з трьох випадків базові класифікатори мають тенденцію до перенавчання (overfitting). Проте між собою результати не відрізняються аж так що б дуже суттєво. Кореляція точності складає близько 10%, що для даних методів не є дуже суттєво.

Для досягнення цілей поставленого завдання дисертації а також опираючись на результати попереднього дослідження проведеного в попередньому розділі було вирішено використовувати класифікатор заснований на деревах, а саме Gradient Boosting Tree. Існує дві основні переваги вибору саме цього типу класифікатора є бажання забезпечити можливість визначення емоцій для текстів з різних категорій, а також обробка в режимі реального часу, тобто потокова обробка тестових повідомлень, самий процес навчання класифікатора буде виконуватись окремо і незалежно від основної частини дисертації, хоча і становить значну її частину.

3.6 Gradient Boosting Tree

Метод градієнтного бустінгу - це техніка машинного навчання для задач класифікації і регресії, яка будує модель передбачення в формі ансамблю слабких пророкуючих моделей, зазвичай дерев рішень. Мета будь-якого

алгоритму навчання з учителем - визначити функцію втрат і мінімізувати її. Давайте звернемося до математики. Нехай, наприклад, в якості функції втрат буде середньоквадратична функція (MSE):

$$Lose = MSE = \sum(y_i - y_i^p)\,2\ . \quad (3.4)$$

Ми хочемо, що побудувати наші передбачення таким чином, щоб MSE була мінімальна. Використовуючи градієнтний спуск і оновлюючи передбачення, засновані на швидкості навчання (learning rate), шукаємо значення, на яких MSE мінімальна.

$$y_i^p = y_i^p + a * b \sum(y_i - y_i^p)\,2/2by_i^p\ . \quad (3.5)$$

Отже, ми просто оновлюємо передбачення таким чином, що сума наших відхилень наближалася до нуля і передбачені значення були близькі до реальних.

3.7 Оцінка якості роботи класифікатора

Існує два підходи до оцінки якості класифікації. перший - це порівняння класифікаторів між собою, другий - абсолютна оцінка якості. Взагалі кажучи, досить складно оцінити якість класифікації. Часто навіть досвідчені експерти не можуть визначити до якої категорії віднести конкретний документ. Найбільш поширена система (метрика) оцінки якості класифікації включає в себе оцінку двох характеристик класифікатора - точність і повноту.

Точність системи в межах класу - це частка документів, дійсно належать даному класу щодо всіх документів, які система віднесла до цього класу. Повнота системи - це частка знайдених класифікатором документів, що належать класу щодо всіх документів цього класу в тестовій вибірці.

Ці значення легко розрахувати на підставі таблиці контингентності, яка складається в такий спосіб:

Таблиця 3.1- Оцінка роботи класифікатора

Категорія I	Експертна оцінка	
	Позитивна	Негативна
Оцінка системи	TP	FP
	FN	TN

У таблиці міститься інформація про те, скільки разів система прийняла вірне і скільки разів невірне рішення за документами заданого класу. А саме:

- TP - істино-позитивне рішення;
- TN – істино-негативне рішення;
- FP - помилково-позитивне рішення;
- FN – помилково-негативне рішення.

На основі вищеописаних величин проводиться підрахунок метрик, які не тільки оцінити якість роботи класифікатора, а й дозволяють порівняти зробити порівняння класифікаторів між собою. В даній роботі були використані наступні метрики, які є найбільш поширеними і часто використовуваними.

Accuracy (Повна точність).

Частка коректно класифікованих документів від загального числа документів, що подаються на вхід класифікатору.

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}. \quad (3.6)$$

Precision (точність).

Для позитивних прикладів - частка документів, які є позитивними від загального числа прикладів, класифікованих як позитивні. Іншими словами, кількість позитивних документів серед всіх документів, які класифікатор вважає позитивними.

$$Precision = \frac{TP}{TP+FP}. \quad (3.7)$$

Для негативних документів - частка документів, які є негативними від загального числа прикладів, класифікованих як негативні.

Recall (повнота).

Для позитивних прикладів - частка правильно класифікованих позитивних прикладів від загального числа позитивних прикладів. Іншими словами, це частка дійсно позитивних документів з усіх документів, розпізнаних як позитивні.

$$\text{Recall} = \frac{TP}{TP+FN}. \quad (3.8)$$

Для негативних прикладів - частка правильно класифікованих негативних прикладів від загального числа негативних прикладів, іншими

словами, це частка дійсно негативних документів з усіх документів, розпізнаних як негативні.

F – measure (F - міра)

Міра, що комбінує точність і повноту.

$$F = \frac{2*Precision*Recall}{Precision+Recall} \text{ . (3.9)}$$

Macro_P[2]

Пусть $C = \{c1 \ldots c_{[c]}\}$ множина всіх класів (категорій) даної задачі класифікації. Тоді міра Macro_P - є середнє арифметичне величин Precision для всіх класів з множини С:

$$\text{Macro\_P} = \frac{1}{|C|} * \sum_{\forall i: c_i \in C} \text{precision(ci)} \text{ . (3.10)}$$

Macro_R[2]

Аналогічним чином вводиться величина *Macro*_R. Міра Macro_R – це середнє арифметичне величин *Recall* для всіх класів з множини С.

$$\text{Macro\_R} = \frac{1}{|C|} * \sum_{\forall i: c_i \in C} r\text{ecall(ci)} \text{ . (3.11)}$$

Висновки за розділом

В даному розділі поставлено та наведено математичні характеристики поставлених задач дисертації, а саме, обробка потоку даних, визначення емоційної складової повідомлення, а також визначення характеристик потоку.

На основі порівняння якості класифікації простих моделей реалізованих окремо на одному з трьох методів машинного навчання: Gradient Boosting Tree, SVM (метод опорних векторів) та Logistic Regression (логістична регресія) було визначено Gradient Boosting Tree як кращий метод. Його і було взято для подальшого дослідження для вирішення однієї із задач.

Також було наведено методи оцінювання роботи класифікатора, на основі стандартних величин, таких як точність, повнота, повна точність, і деяких агрегованих, таких як F-міра, Macro_P, Macro_M.

На основі поставленої задачі, було описано процес збору набору даних, для вирішення задачі, наведено його характеристику і приклад елементів з набору даних. Набір даних складається з близько 4 тисяч елементів, зібраних на просторах інтернету, кожен елемент написано українською мовою.

4 ОПИС ПРОГРАМНОГО ТА ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ

Цей розділ присвячений розробці програмного забезпечення з мінімальним рівнем набором функцій, які можна використовувати як окремий блок для інтеграції в інших системах або розвиваються і стають повноцінним продуктом, який можна і потрібно інтегрувати в інші продукти, де його можна використовувати як підсистему.

Далі наведенні обґрунтування у виборі засобів розробки для кожної із підсистеми, також описано інфраструктуру яка була використана і піднята для виконання поставленого завдання.

4.1 Аналіз вимог

Розробка будь-якої системи та продукту, включаючи будь-який програмний продукт чи наукове дослідження, починається з формулювання вимог до системи чи підсистеми яку буде отримано на виході.

Вимоги до проектованої системи являють собою сукупність тверджень про атрибути, властивості, або якості програмної системи чи підсистеми, що підлягають реалізації в майбутньому.

Якщо говорити конкретно про нашу систему, то слід прояснити що вона повинна мати можливість вирішувати завдання, поставлені на початку цієї дисертації.

На вхід до системи подається потік текстових даних, який береться з месенджера telegram, для прикладу, на виході система повинна віддати метрики потоку даних, а також сентимент кожного повідомлення з потоку.

Для кращого формулювання вимог розділимо їх на функціональні та нефункціональні. До перших віднесемо ті, що стосуються конкретно набору функцій, а саме, формування потоку даних, алгоритми визначення сентименту, мови повідомлення, візуалізацію. Тобто алгоритмічна складова, розгортання, функції та поведінку системи. До інших віднесемо бізнес складову системи, тобто зовнішній вигляд, UI та UX. Дане дослідження має більш науковий характер, тому набір вимог до системи буде мінімальним.

До категорії функціональних вимог віднесемо наступні:

- можливість зчитувати повідомлення з месенджера telegram;
- можливість отримувати повідомлення з різних каналів/чатів/ груп;
- можливість визначати мову повідомлення;
- можливість зберігати проміжні стани процесу;
- можливість візуалізації потоку за різними метриками;
- можливість реагувати на зміну характеристик потоку.

До ж не функціональних вимог віднесемо наступні:

- можливість запуску в браузерах chrome, firefox, safari;
- програма повинна зручний інтерфейс для візуалізаці;
- програма повинна реагувати на різку зміну настрою потоку;
- програма повинна мати інтеграцію з месенджером телеграм.

4.2 Docker та docker-compose

Однією з вимог до сучасних систем є можливість швидкого та безболісного розгортання розгортання, швидкого масштабування, перенесення даних з одного серверу на інший. Для цього є декілька видів рішень, наприклад можна написати bash/powershell скрипт який би розгортав ваше рішення, виконував масштабування та все інше в автоматичному режимі, але ж що робити, якщо наприклад у вас декілька різних частин кластера, які вимагають запуску на різних ос, наприклад веб-сервер написаний на java, і для його роботи потрібна Linux Ubuntu, а база даних MSSQL, для якої потрібний Windows Server, можна звичайно використовувати різні машини для цього всього. А якщо таких мікро-сервісів десятки чи сотні. Тобто виникає питання ізольованості між нодами кластеру, і при цьому незалежній і різній роботі кожної із них. Для цього нам на допомогу приходять такі інструменти як Docker для docker-compose.

Docker - це інструментарій для управління окремими контейнерами.

Docker доповнює інструментарій LXC Toolkit, створивши більш досконалий API, який дозволяє керувати контейнерами на рівні окремих процесів. Зокрема, Docker дозволяє запускати довільні режимі ізоляції не турбуючись про вміст контейнера, а потім переміщувати та клонувати контейнери, створені для цих процесів, на інші сервери, усі роботи зі створення, обслуговування та обслуговування контейнерів він бере на себе.

Docker - це інструмент, призначений для спрощення створення, розгортання та запуску програм за допомогою контейнерів. Контейнери дозволяють розробнику пакувати додаток з усіма необхідними йому частинами, такими як бібліотеки та інші залежності, і доставляти все це як один пакет. Тим самим, завдяки контейнеру, розробник може бути впевнений,

що програма запуститься на будь-якій іншій машині незалежно від будь-яких налаштованих параметрів, які можуть мати машини, які можуть відрізнятися від машини, що використовується для написання та тестування коду.

Docker дещо схожий на віртуальну машину. Але на відміну від віртуальної машини, замість створення повноцінної операційної системи, Docker дозволяє програмам використовувати те саме ядро Linux, як і потрібна йому операційна система, і лише заряджає програми тим, що вже не працює на хост-комп'ютері. Це підвищує продуктивність і зменшує розмір програми.

Docker - це інструмент, призначений приносити користь як розробникам, так і системним адміністраторам, що робить його частиною багатьох наборів інструментів DevOps (розробники + процеси). Для розробників це означає, що вони можуть зосередитись на написанні коду, не турбуючись про те, на якій системі він в кінцевому підсумку буде працювати. Це також дозволяє їм розпочати роботу з однією з тисяч програм, які вже розроблені для запуску в контейнері Docker в рамках програми. Для оперативного персоналу Docker забезпечує гнучкість і потенційно зменшує кількість необхідних систем через їх невеликий розмір і невелике навантаження.

Під час вирішення завдання розробки Docker використовувався широко, для ізоляції Кожної підсистеми та їх частин одна від одної, це перш за все, також для запуску всіє системи на одній машині, оскільки різні компоненти вимагають різні операційні системи, Alpline Linux та Ubuntu Linux, а також для запуску системи в цілому на віддаленому середовищі.

4.3 Архітектура програмного забезпечення

У даному розділі описано підсистеми програмного продукту розробленого у процесі вирішення задачі поставленої на початку даного дослідження. Кожна підсистема розроблена максимально атомарно, що б у разі потреби її можна було замінити на інше рішення, коли наприклад потрібно замінити на щось надійніше/швидше, а також ізольовано одна від одної, тобто системи не можуть навмисне змінити поведінку одна однієї, лише виконують спілкування одна з іншою через вузько сформований контракт.

Кожна з підсистем запускається в окремому docker-кластері, які спілкуються використовуючи спільну приватну мережу, тобто отримати доступ до елементів підсистеми неможливо. Отримати доступ можна тільки по визначених портах та протоколах, використовуючи визначений набір контрактів. Тобто не можна оминувши перші дві системи, відправити повідомлення на аналіз.

Також в елементах системи налаштоване авто-масштабування що дозволяє уникнути «неочікуваній» просадок по швидкості та надійності рішення, також завдяки цьому, якщо елементи системи/підсистеми виходить з ладу, піднімаєте інший, який заміняє попередній, і система працює без будь-яких затримок.

В процесі дослідження усю архітектуру рішення було поділено на 4 підсистеми:

- підсистема збору повідомлень;
- підсистема транспортування повідомлень;
- підсистема аналізу повідомлень;
- підсистема зберігання результату та візуалізації.

На рисунку 4.1 зображена архітектура системи в цілому, включаючи компоненти підсистем.

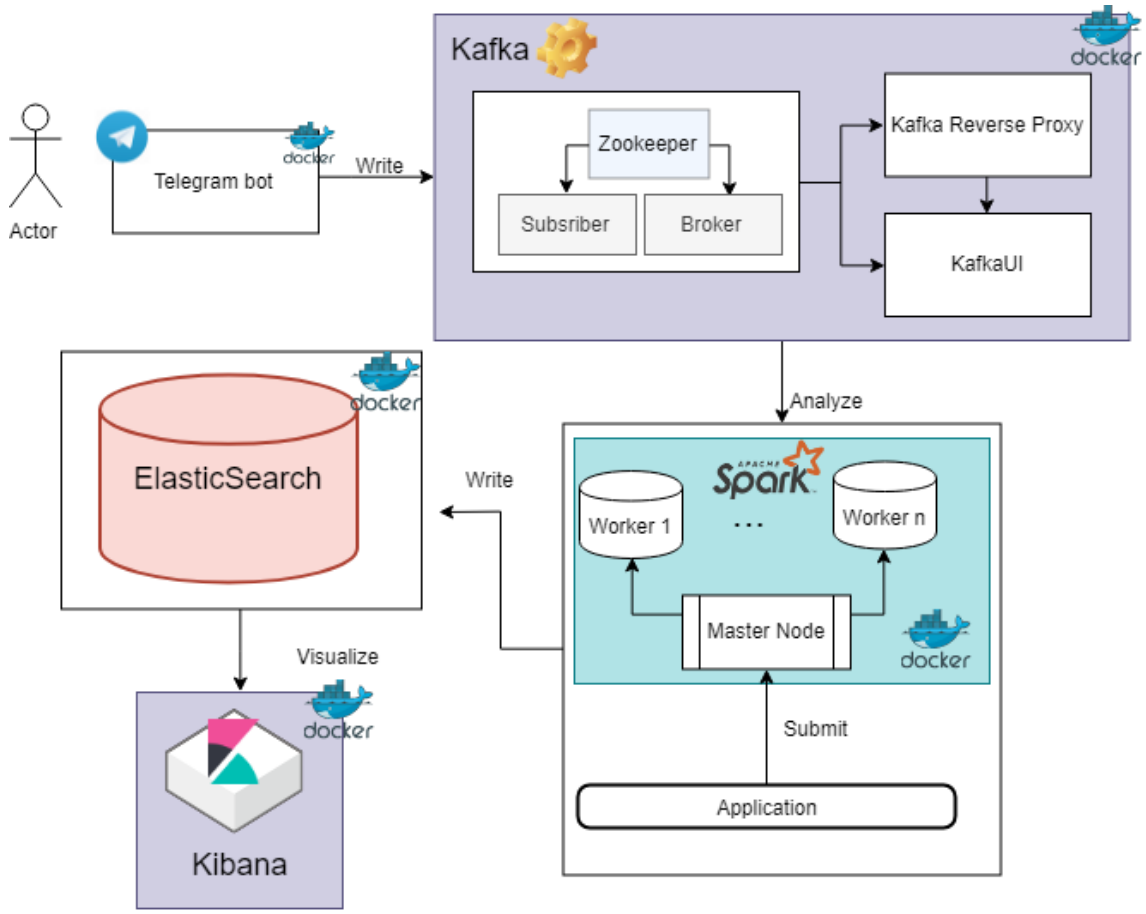


Рисунок 4.1 – Схема потоків даних

4.4 Підсистема збору повідомлень

Головна мета даної підсистеми збирати повідомлення з месенджеру телеграм, які потрібно проаналізувати, вичитувати всі необхідні дані для аналізу, включаючи інформацію про користувача. Далі формується потік даних та відправляється в наступну підсистему.

Для розробки підсистеми було обрано мову програмування JavaScript, так як найбільш підходящу для виконання поставленого завдання.

JavaScript – мова програмування, яка розроблялася як браузерна для додавання інтерактивності в сторінки, зараз використовується в сучасних web фрейморках, для написання web серверів, ботів, настільних та мобільних програмних засобів.

Для створення бота, який можна використати, з меседжером телеграм, використовувався фреймворк для написання програмних засобів такого типу.

Telegraf.js - це досить проста бібліотека, яка спрощує безліч набридливих завдань, з якими стикаються під час розробки ботів.

Бот слухає всі повідомлення з усіх чатів, груп та каналів у які він доданий, пре обробляє їх, згідно визначеного контракту та відправляє далі. Контракти, за яким бот спілкується з іншими підсистемами, описаний нижче. Спілкування виконується у форматі JSON, який на момент написання дисертації, є найпоширенішим та найбільш-популярним.

Таблиця 4.1 – Опис контракту

Назва	Опис
message	текст повідомлення
user_full_name	прізвище, ім’я користувача який відправив повідомлення
from_username	ім’я користувача
chat_title	назва чату
chat_name	ім'я чату
chat_id	ідентифікатор чату
date	дата відправки повідомлення

Всі вказані вище поля потрібні для правильного аналізу повідомлення, та візуалізації, групування для визначення характеристик потоку. В залежності від виду чату, це може ім’я користувача, назва групи чи каналу.

4.5 Підсистема транспортування повідомлень

Дана підсистема є наступною в ланцюгу підсистем для аналізу потоків даних, головна її суть полягає в транспортуванні повідомлень з підсистеми збору повідомлень в підсистему аналізу повідомлень.

Підсистема складається з декількох модулів, а саме:

- черга повідомлень kafka;
- kafka reverse proxy;
- kafka ui;
- zookeeper.

Черга повідомлень kafka використовується для побудови потоків даних і стрімінгових сервісів в режимі реального часу. Kafka горизонтально масштабована, стійка до відмов і використовується в тисячах компаній.

Kafka складається з скінченої кількості нод, тобто процесів як взаємодіють між собою. Вони поділяються на два види: master-node головна нода, яка відповідає за розподілення завдань на дочірні. Та worker-node, дочірня нода, яка в свою чергу виконує транспортування. Кількість дочірніх нод змінюється в процесі роботи підсистеми в залежності від навантаження яка надходить на вхід до підсистеми. Це все відбувається автоматично, завдяки механізмам, які надає нам docker-compose.

Kafka – це черга повідомлень типу publish/subscribe тобто всі повідомлення які надходять на вхід у топік (publish) автоматично пересилаються всім існуючим підписникам (subscribe). Топік, буквально звучить як тема, це місце обробки повідомлень куди вони надсилаються та передаються далі кінцевими підписникам.

У випадку виконання завдання дисертації, нам потрібна черга з конфігурацією 1:1, тобто один продюсер, підсистема збору повідомлень та один консюмер, підсистема обробки повідомлень.

Також, для полегшення моніторингу даної підсистеми, до неї включено два сервіси, kafka-reverse-proxy та kafka-ui. Перший потрібний для отримання доступу до черги через протокол доступу http/https, тобто обгортка навколо існуючої черги повідомлень. Kafka-ui являє собою web інтерфейс, який можна відкрити у браузері. Він має весь необхідний інструмент для моніторинг, маніпуляції та управління чергами повідомлень.

Приклад інтерфейсу kafka-ui наведений на рисунку нижче.

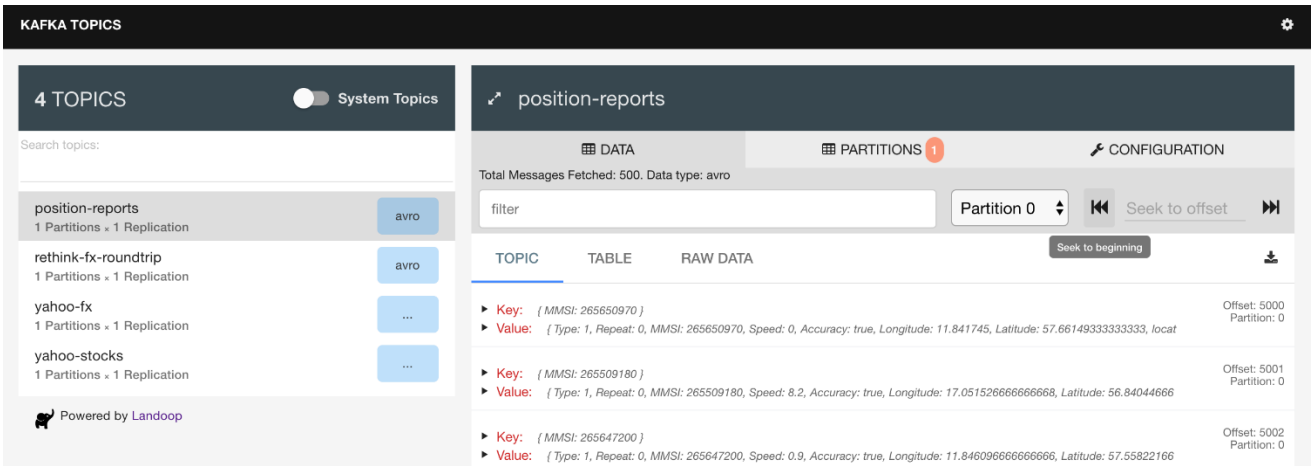


Рисунок 4.2 – Приклад інтерфейсу сервісу kafka-ui

Також для управління всім станом підсистеми, збереження конфігурації про черги, топіки та повідомлення використовується сервіс zoo-keeper.

ZooKeeper - це централізований сервіс для підтримки інформації про конфігурацію, іменування, забезпечення розподіленої синхронізації та надання для груп послуг.

Отже, підсумуємо, під час розробки підсистеми було запущено та налаштовано чергу повідомлень kafka, з необхідними інструментами для моніторингу та збереження конфігурації та стану між запусками.

Також створений топік з конфігурацією **1:1,** а також інтегровано його з підсистемою збору повідомлень.

Кожен сервіс підсистеми запускається в окремому docker контейнері, а також завдяки інструментам docker-compose налаштовано оркестрацію та авто-балансування навантаження в залежності від кількості повідомлень.

4.6 Підсистема аналізу повідомлень

Дана підсистем найскладніша з усіх написаних під час виконання даного дослідження, адже її основне завдання це аналіз тих повідомлень які надходять на вхід до неї у вигляді потоку даних. Як характеристика потоку даних система визначає мову та емоційну складову кожного з повідомлень. Приблизний алгоритм визначення характеристик потоку наведений нижче.

а) Отримати повідомлення.

б) Визначити мову повідомлення.

в) Якщо мова повідомлення – українська використати стратегію визначення емоцій, розроблену в процесі дослідження, алгоритм якої наведений нижче.

г) Якщо мова повідомлення англійська – використати стенфордський алгоритм визначення емоцій в тексті.

д) Якщо мова повідомлення не українська чи не англійська, поставити емоційне значення як нейтральне (neutral 0.5).

е) Перед обробити текстове повідомлення, відсіяти зайві складові, привести слова до нормальної форми.

ж) Визначити емоційне складову текстового повідомлення, використовуючи відповідну стратегію визначену вище.

з) Передати результат за визначеним контрактом в наступну підсистему.

Виходячи з дослідження проведеного в розділі 1, для потокової обробки повідомлень в даному випадку краще за все використовувати Apache Spark, через переваги, які наведені в результат зазначеного розділу.

Мову програмування, для цієї написання підсистеми обрану Scala, через ряд переваг, а саме:

- spark написаний на Scala, тому є можливість використовувати оригінальні api без адаптерів;
- функціональне програмування, надає гнучкість у написанні та тестуванні коді на відмінну від ООП;
- можливість швидкої збірки та компіляції коду завдяки sbt.

Перш за все завдяки першому пункту і обрану цю мову програмування.

Scala поєднує об'єктно-орієнтоване та функціональне програмування однією стислою мовою високого рівня. Статичні типи Scala допомагають уникнути помилок у складних програмах, а її JVM та JavaScript під час виконання дозволяють створювати високопродуктивні системи з легким доступом до величезних бібліотек.

Дана підсистема складається з двох сервісів, які взаємодіють один з одним утворюючи підсистему.

Перший сервіс, це кластер Apache Spark, який складається з однієї master-node, тобто головної ноди, яка займається розподілом завдань та навантаження між дочірніми нодами (worker-node). Завдяки відомому нам функціоналу docker-compose, а саме авто-балансування навантаження, їх кількість змінна, і збільшується чи зменшується в залежності від навантаження, яке надходить на підсистему. Кожна нода запускається в окремому ізольованому контейнері.

Як базове зображення (docker image) використається зібране зображення, організації BigData Europe, яке поширюється під відкритою ліцензією Масачусетського Інституту MIT. Кожне зображення використовує alpine linux як операційну систему, яка трішки відрізняється від звичної всім Ubuntu, але краще підходить під завдання такого типу.

Інший сервіс – це саме програмне забезпечення, у вигляді завдання яке виконується на Spark.

Сервіс використовує модель для машинного навчання, навчену на зібраному дата сеті та скомпільована з використання бібліотеки Spark.ML.

Процес навчання моделі описаний нижче.

а) Завантажити дата сет в пам’ять.
б) Визначити кількість позитивних/негативних повідомлень.
в) Форматувати повідомлення в формат підходящий для навчання.
г) Виділити валідаційну вибірку.
д) Сформувати стратегію навчання.
е) Навчити модель.
ж) Провалідувати навчену модель.
з) Зберегти у файл.

Після декількох тестів на різних параметрах стратегії навчання, було визначено найоптимальніші із них які показали найточніший результат на валідаційній вибірці:

- кількість класів: 2 (позитивне чи негативне повідомлення);
- кількість епох: 30;
- глибина дерева: 6.

Після навчання моделі вона використовується в процесі визначення емоційних характеристик потоку, алгоритм якого наведений на початку підрозділу.

Після виконання алгоритму визначення емоційної складової потоку, дані надсилаються в наступну підсистему яка збереже результат для майбутнього аналізу і візуалізує його.

4.7 Підсистема зберігання результату та візуалізації.

Головна мета підсистеми зберегти результат виконання в нереляційну базу даних за для подальшого використання. Підсистема, як і попередня складається з двох окремих сервісів, які взаємодіють між собою.

Перший сервіс – це нереляційна база даних, Elastic Search.

Бази даних NoSQL добре підходять для багатьох сучасних додатків, таких як мобільні, ігрові, інтернет-програми, коли потрібні гнучкі масштабовані бази даних з високою продуктивністю та широкою функціональністю, що може забезпечити максимальну зручність використання.

ElasticSearch, як сховище даних, був обраний не випадково, оскільки він дуже добре взаємодіє з наступним сервісом, Kibana. Можна сказати що ці сервіси створені один для одного. ES має наступні переваги:

- гнучкість. Як правило, бази даних NoSQL пропонують гнучкі схеми, що дозволяє прискорити розробку та поетапну реалізацію;
- масштабованість. Бази даних NoSQL розраховані на масштабування з використанням розподілених кластерів апаратного забезпечення, а не шляхом додавання дорогих надійних серверів;
- висока продуктивність. Бази даних NoSQL оптимізовані для конкретних моделей даних;

- широкі функціональні можливості. Бази даних NoSQL надають API і типи даних з широкою функціональністю, які спеціально розроблені для відповідних моделей даних.

Дані в базі даних у одному єдиному індексі, який створюється автоматично при першому записі даних, індекс показує джерело даних звідки вони були отриманні, це означає що в подальшому можлива підтримка ще одного джерела даних, окрім телеграм.

Структура даних в індексі наведена нижче, з описом:

- lang – визначенна мова;
- sentiment - значення сентименту в повідомленні;
- text – повідомлення;
- chat_id – ідентифікатор чату;
- chat_name – ім’я чату;
- author_full_name – повне ім’я автора або нік;
- date – дата створення повідомлення;
- processing_date – дата визначення сентименту.

Після запису даних у нереляційну базу даних, вони автоматично потрапляють у сервіс візуалізації – Kibana. Оскільки вона автоматично кожні n-секунд відвантажує данні з вказаного індексу, що дозволяє виконувати візуалізацію потоку в режимі реального часу.

Kibana - це служба візуалізації даних Elasticsearch та навігації по Elastic Stack. Вона допомагає створити панелі приладів, налаштувати форму візуалізації, сформувати інтерактивні графіки, навіть представити геодані, проаналізувати взаємозв’язки та вивчити аномалії за допомогою машинного навчання.

Завдяки гнучкому інтерфейсу останнього сервісу у ньому було створені два види графіків, а саме графік для відображенню емоційної складової потоку в загальному, та по специфічному чаті.

4.8 Інструкція користувача

Для того щоб почати використовувати розроблену систему, кінцевому користувачеві потрібно мати:

- акаунт в месенджері телеграм;

- доступ до мережі інтернет;

- встановлений браузер.

Для початку, користувачеві потрібно додати в свій діалог/групу/канал бота, @Notify4Me_bot, скріншот з яким наведений нижче. Бот буде автоматично брати відправлене йому повідомлення та відправляти на аналіз.

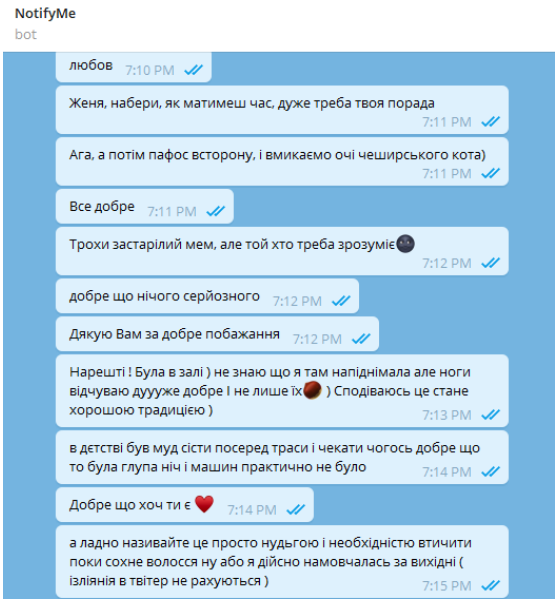


Рисунок 4.3 - Скріншот діалогу з ботом

Після аналізу, результати будуть відправлені на вебсервер для зберігання, а результат можна буде побачити в браузері відкривши посилання на нього.

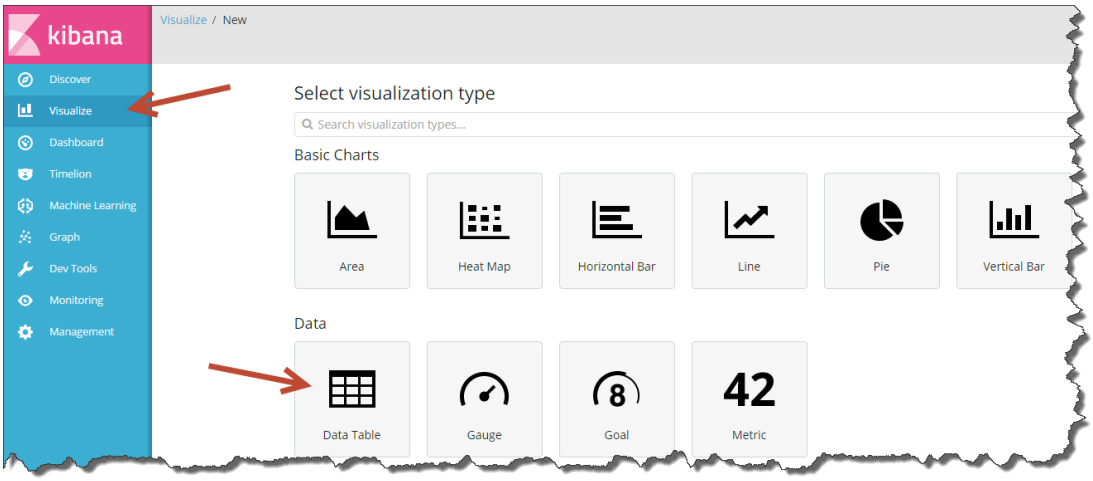


Рисунок 4.43 - Приклад графічного інтерфейсу

Висновки до розділу

У данному розділі описано та проаналізовано вимоги до розроблювальної системи, як функціональні та і нефункціональні. Також було описано інструментарій для розробки програмного забезпечення та аргументовано його вибір.

Наведено опис кожної підписсистеми для програмного продукту, з деталями реалізації та розгортання.

Система поділяється на 4 підсистеми, кожна з яких може розгортатися окремо та ізолювано від інших і може бути замінена на будь-яку іншу реалізацію, зберігаючи контракт.

5 РЕЗУЛЬТАТИ ЧИСЛОВИХ ЕКСПЕРИМЕНТІВ

5.1 Апаратне забезпечення для проведення експерименту

Для тестування використовувався персональний комп’ютер:

- 4 ядерний CPU: Intel(R) Core(TM) i7-6500 CPU @ 3.20GHz;
- GPU: Nvidia GeForce GTX 960 4Gb;
- RAM: 16Gb Kingston DDR3 SDRAM.

Для порівняння виконання алгоритмів використовується тестовий набір даних близько 4 тисяч текстових повідомлень користувачів зібраних з різних соціальних мереж.

Для тестування поділимо наш набір даних на дві частини, одна – на якій модель буде навчатися складатиме близько 70% набору даних, інша - валідаційна, 30%. Даний розподіл якнайкраще підходить для тестування завдань такого типу.

5.2 Визначення точності виконання алгоритму Gradient Boosting

Gradient Boosting Tree – алгоритм заснований на деревах, і має певну конфігурацію для навчання.

- Кількість ітерацій.
- Глибина дерева.
- Кількість класів.

Виконаємо декілька тестів на різних параметрах для визначення найкращої та найточніших параметрів конфігурації.

Таблиця 5.1 - Тестування GBT

Кількість ітерацій	Глибина дерева	Час виконання, ms	Помилка на тестовому наборі даних %	Помилка на валідаційному наборі даних %
20	4	42	21.13	35.11
20	6	41	20.85	29.45
20	8	44	20.35	28.87
30	4	45	19.02	27.63
30	6	40	15.17	25.99
30	8	38	17.13	28.63
40	4	41	20.37	31.12
40	6	42	21.03	33.23
40	8	50	21.56	34.69

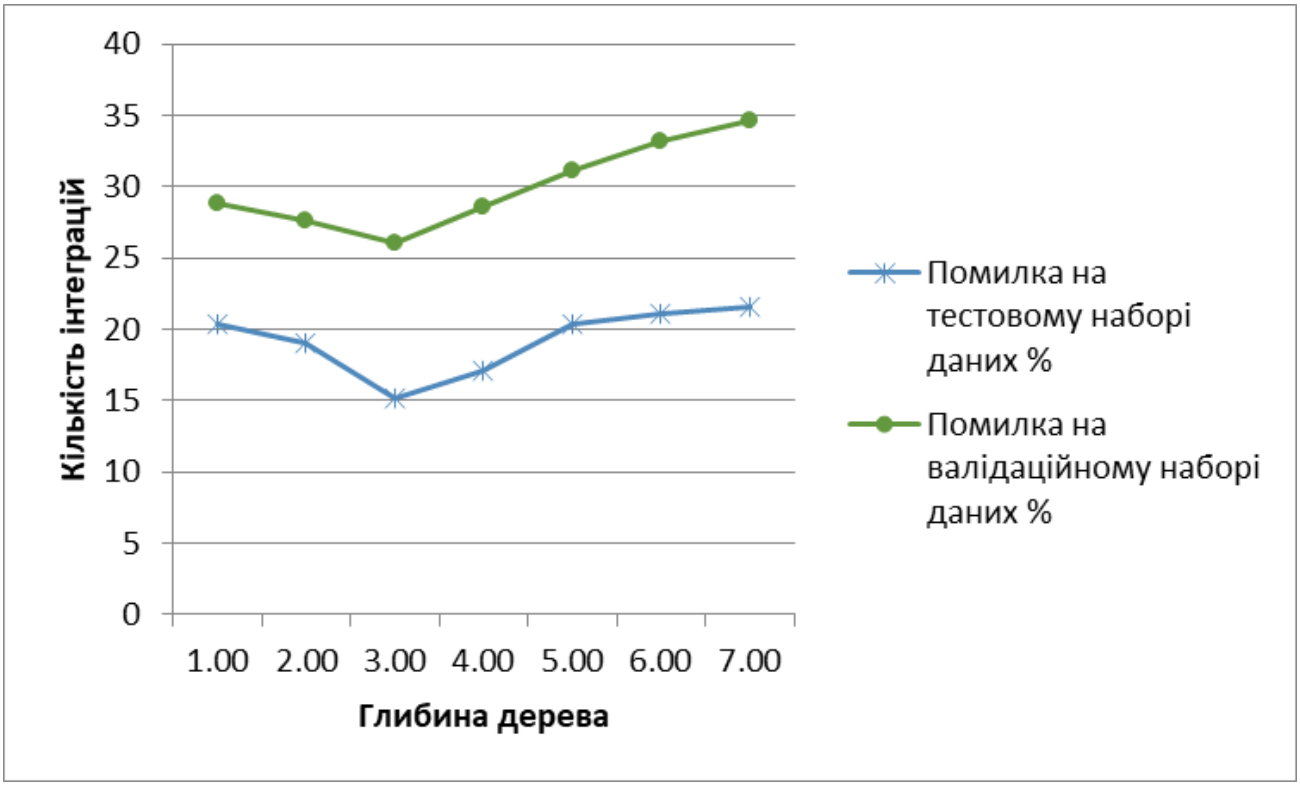


Рисунок 5.1 - Візуалізація результату

На графіку чітко видно залежність точності виконання від вхідної моделі даних.

Як бачимо з результатів тестування, найкращі параметри для навчання класифікатора – це кількість ітерацій – 30, глибина дерева – 6. Запам’ятаємо ці параметри та виконаємо тестування ще одним методом для порівняння результатів і вибору найкращого класифікатора.

5.3 Визначення точності алгоритмом Random Forest

Random Forest алгоритм також заснований на деревах, який має дещо інший набір конфігурації навчання моделі.

Конфігурація складається з наступних параметрів:

- кількість дерев;
- кількість класів.

Кількість класів у нашому досліджені завжди дорівнює двум, позитивний чи негативний.

Виконаємо декілька тестів на різних параметрах для визначення найкращої та найточніших параметрів конфігурації.

Таблиця 5.2 - Тестування Random Forest

Кількість дерев	Час виконання, ms	Помилка на тестовому наборі даних %	Помилка на валідаційному наборі даних %
2	64	21.62	31.74
5	61	20.12	28.89
10	65	19.89	28.63
13	56	20.39	29.32
15	69	20.96	30.64
20	63	22.48	32.89

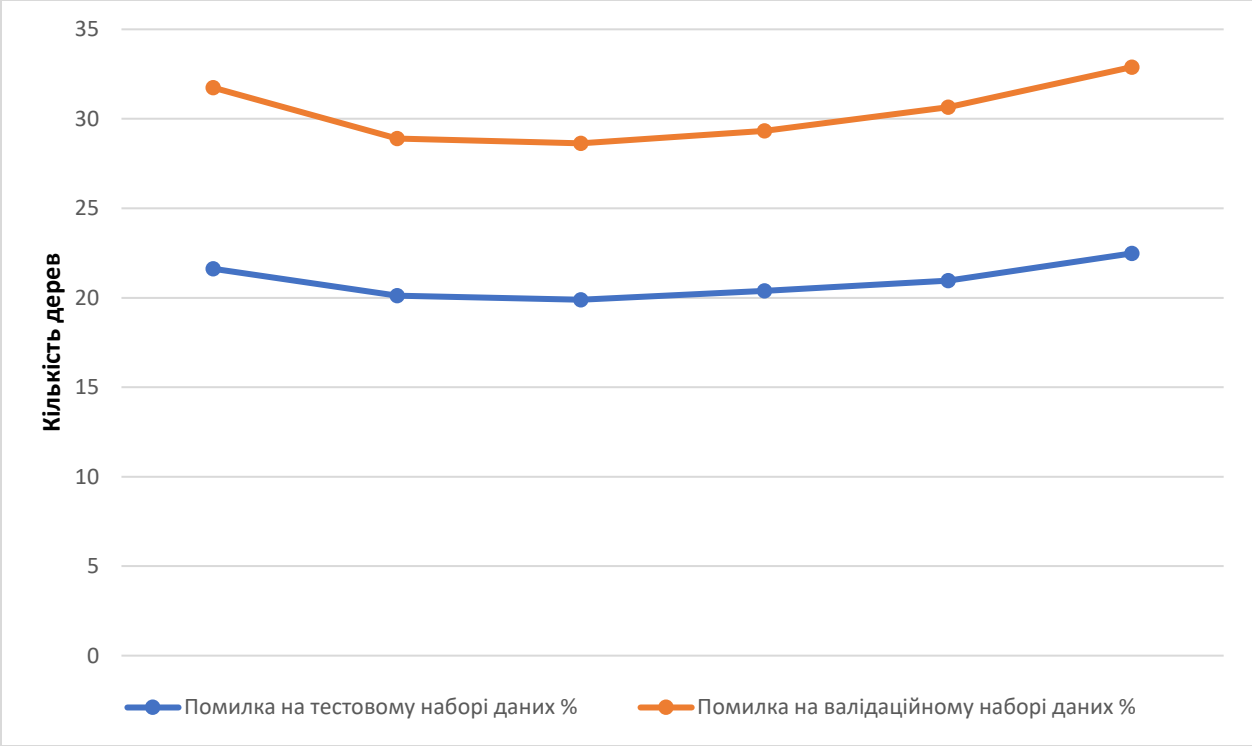


Рисунок 5.2 - Візуалізація результату

Отже, як бачимо з результатів тестування, найкращий показник точності спостерігається на конфігурації в 10 дерев. Приблизна точність на цьому показникові – 28.63 на валідаційному наборі даних та 19.89 на тестовому.

5.4 Порівняння виконання Gradient Boosting та Random Forest

Після виконання тестування Gradient Boosting та Random Forest алгоритмів, потрібно виконати підсумуючу візуалізцію результатів виконання цих графіків.

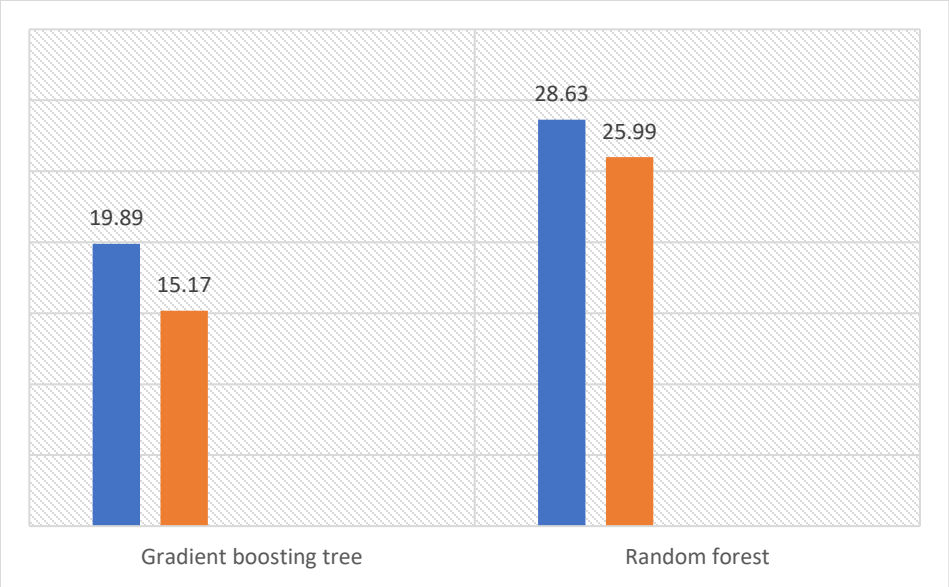


Рисунок 5.3 - Порівняння результатів для різних методів

Висновки до розділу

У даному розділі порівняно результат виконання алгоритму Gradient Boosting Tree, на варіаційному наборі даних, порівняно результати кожної з конфігурації та обрану найоптимальніший з варіантів.

Також було проаналізовано алгоритм Random forest, на різних наборах конфігурацій, та визначено найоптимальнішу з них.

Було визначено, що алгоритм Gradient Boosting Tree працює точніше на визначеному наборі даних, зібраного в процесі виконання дослідження, визначено найоптимальнішу конфігурацію для нього, що б використати її в процесі наступного дослідження.

6 РОЗРОБЛЕННЯ СТАРТАП ПРОЕКТУ

6.1 Опис ідеї проекту

У даному розділі буде проведено аналіз стартап проекту, для визначення можливості виходу на ринок розроблюваного продукту, та можливості конкурувати з вже існуючими продуктами.

Призначенням проекту є реалізація можливості отримання емоційної оцінки повідомлень для будь-якої людини, відреагувати на різку зміну емоційної складової тексту.

Сутністю розробки є створення інструменту сентимент аналізу, призначений для визначення емоційної складової текстових повідомлень в режимі реального часу, реакцію різкої зміни емоційної складової та її візуалізацію.

Цільовою аудиторією для використання цього продукту є аудиторії соціальних мереж, месенджерів, адміністратори або модератори груп чи каналів.

Основною вигодою використання цього продукту допоможе швидко та ефективно визначати емоції в повідомленнях, для того що б реагувати відповідним чином на те чи інше повідомлення в каналі.

Для отримання цілісного уявлення про зміст ідеї, а також базові можливі потенційні ринки збуту, в межах яких потрібно шукати групи клієнтів, які в використовували даний продукт, опишемо зміст ідеї стартап проекту, можливі напрямки застосування, а також вигоди для користувача у таблиці 4.1.

Таблиця 6.1 - Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Система сентимент аналізу повідомлень в режимі реального часу з можливістю реакції на зміну емоцій текстового потоку та візуалцізацією	Медіатехнології	Автоматизований аналіз емоцій в режимі реального часу, для каналів та чатів в месенджерах та інших джерелах.Система може аналізувати дані з різних джерел інформації та візуалізувати емоційну складову потоку тестових повідомлень.

Отже, ідея створення системи автоматизованого аналізу емоційної складової тексту в режимі реального часу є доволі актуально враховуючи популярність меседжерів чи програм для обміну повідомлень.

З огляду на те що з кожним роком люди сприймають та продукують все більше інформації з електронних джерел, розроблена система може інтегуватися з найпопулярнішими меседжерами та соціальними мережами.

Наступним кроком - це проведення аналізу потенційних техніко-економічних переваг ідеї, у порівнянні із ідеями, що пропонують конкуренти: визначити перелік економічних та технічних властивостей та характеристик ідеї; визначити конкурентів, що вже існують на ринку, та провести збір

інформації щодо значень технічних та економічних показників для ідеї власного проекту та конкурентних проектів відповідно до переліку, визначеного вище; провести аналіз та порівняння показників: для власної ідеї визначити показники, що мають

а) гірші значення (W, слабкі);

б) аналогічні (N, нейтральні) значення;

в) кращі значення (S, сильні) (табл. 6.2).

Після детального пошуку у мережі інтернет інструментів подібних до запропонованої системи був знайдений один конкурент - український стартап leegle.

У таблиці 6.2 наведемо порівняльний аналіз сильних, слабких та нейтральних характеристик ідеї проекти у порівнянні із зазначеними конкурентами.

Таблиця 6.2 - Визначення сильних, слабких та нейтральних характеристик ідеї проекту

Ідея	**(потенційні) товари/концепції конкурентів**		**W (слабка сторона)**	**N (нейтральна сторона)**	**S (сильна сторона)**
	Мій проект	**Leegle**			
Автоматизований аналіз тексту на предмет застосування прийомів впливу	+	+		+	
Виділення елементів тексту що визначені як патерни психологічного впливу	-	-	-		
Обмеження за обсягом введених даних	+	-			+

Продовження таблиці 6.2

Інтеграція системи з різними операційними системами(Android, iOS, Smart TV)	+	+		+	
Аналіз популярних електронних ресурсів на предмет використання патернів психологічного впливу	+	-			+

Виходячи з конкретних сильних, слабких та нейтральних якостей ідеї, можна проаналізувати та зрозуміти її конкурентоспроможність.

Таким чином, згідно з наведеною вище таблицею, можна зазначити, що проектний проект має ряд переваг перед аналоговими системами (їх конкурентами), а саме: аналіз популярних електронних ресурсів на предмет використання патернів психологічного впливу та відсутність обмеження за обсягом введених даних.

Однак на цьому етапі розвитку не планується включати велику кількість інтеграцій з різними системами через брак часу. Але інтеграція може бути реалізована при необхідності. Крім того, через певні обмеження застосовуваних методів неможливо ідентифікувати певні розділи тексту, які

знайомі із законами психологічного впливу. Але цю функцію конкурент не виконує.

6.2 Технічний аудит ідеї проекту

У наступному розділі ми розглянемо методи, які можуть допомогти вам реалізувати свої ідеї запуску. Зв’язок між ідеєю проекту та технологією, необхідною для її реалізації, показаний у таблиці 6.3.

Таблиця 6.3 - Технологічна здійсненність ідеї проекту

Ідея проекту	**Технології її реалізації**	**Наявність технологій**	**Доступність технологій**
Алгоритм класифікації текстів	Мова програмування Scala та бібліотека MLib	Наявна	Доступна безкоштовно
Створення telegram бота для інтеграції з месенджером telegram	BOT, Мова програмуваня JavaScript та Docker для контейнеризації	Наявна	Доступна безкоштовно
Аналіз емоційної складової повідомлень в режимі реального часу з використанням	Мова програмування scala та відкритий сервіс Spark	Наявна	Доступна безкоштовно

Продовження таблиці 6.3

Візуалізація емоцій в тексті за певний період	Платформа Kibana, та Docker для розгортання	Наявна	Доступна безкоштовно

Як видно з наведеної вище таблиці, проект, що розробляється в рамках дослідження, не вимагає розробки нових технологій, а скоріше використовує існуючі технології що присутні у вільному та відкритому доступі.

6.3 Аналіз ринкових можливостей запуску стартап-проекту

Ми визначимо ринкові можливості, які можна використовувати, коли проект виходить на великий ринок, а також загрози, які можуть завдати шкоди проекту після його виходу на ринок.

Це дозволить запланувати можливі напрямки розвитку проекту, враховуючи ситуацію на ринку. Результати узагальнені в таблиці 6.4.

Таблиця 6.4 — Попередня характеристика потенційного ринку стартап-проекту

Номер пункту	Показники	Характеристика
1	Кількість головних гравців	1
2	Загальний обсяг продаж, грн/ум.од	Використання користувачами що сприймають інформації з електронних ресурсів
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Відсутні
5	Специфічні вимоги до стандартизації та сертифікації	Можливі у певних країнах світу
6	Середня норма рентабельності в галузі (або по ринку), %	Невідома

Оцінюючи потенційний стартовий ринок, стартап-проект показує, що умови виходу на ринок дуже сприятливі, особливо враховуючи, що на ринку є конкуренти, якиі не дуже давно на ринку.

6.4 Розробка маркетингової програми стартапу

У цьому розділі сформуємо маркетингову концепцію товару, що отримає кінцевий споживач.

Кінцевий споживач може отримати доступ як до веб версії продукту, так і до бота в такому месенджері як telegram. Для нових користувачів надається тріал-версія продукту строком до 2 тижнів, для подальшого використання необхідно внести щомісячну сплату. Клієнт може підв’язати свою кредитну картку, і плата буде зніматися автоматично.

Для початку визначимо основні переваги продукту, що розробляється у таблиці 6.5.

Таблиця 6.5 — Основні переваги використання розроблювального продукту

Потреба	Вигода від використання продукту	Ключові переваги перед конкурентами
Перевірка тестових повідомлень на наявність спеціальних лінгвістичних конструкцій, для визначення емоційої складової	Класифікація тестових повідомлень на наявність емоційної складової в тексті	Зручний інтерфейс, можливість використовувати бота для телеграму, автоматична класифікація повідомлень за емоційною складовою

Таким чином, ви можете бачити, що продукт має очевидні переваги перед конкурентами та відповідає потребам користувачів. Наступним кроком є вибір системи продажу. Ми надамо інформацію в таблиці 6.6.

Таблиця 6.6 — Система збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Підпис	Основні функції постачальника: - Розробка - Тестування - Вдосконалення - Збут та підтримка	Перший рівень	Пряма, із пошуком клієнтури

Також розробимо стратегію маркетингових комунікацій, та візьмемо за її основу специфіку поведінки клієнта, та стратегії позиціонування. Стратегія наведена у таблиці 6.7.

Таблиця 6.7 — Стратегія маркетингових комунікацій

Специфіка поведінки цільових клієнтів	Канали комунікації	Ключові позиції для позиціювання	Концепція рекламного звернення
Покращення якості наданих послуг, зменшення їх вартості	Інтернет ресурси, контекстна реклама, прес-релізи	Канал першого рівня	Покращення якості продукту, зручний інтерфейс

6.5 Розробка ринкової стратегії стартапу

Для опису ринкової стратегії, необхідно визначити групи потенційних користувачів (таблиця 6.8).

Таблиця 6.8 — Групи потенційних користувачів продукту

Назва групи	Готовність користування проектом	Приблизний попит в межах групи	Інтенсивність конкуренції	Складність входу до сегменту
Журналісти, працівники сфери медіа	Готові	Високий	Середня	Середня
Аудиторія Інтернету	Готові	Середній	Середня	Середня

Згідно таблиці, основними користувачами цього продукту можуть бути журналісти та медіа-професіонали, які займаються інтернет-виданнями та

аналізують велику кількість інформації з інтернет-джерел. Також можливими користувачами можуть стати звичайні користувачі Інтернету, які хочуть перевіряти інформацію на наявність пропаганди. Саме цим сегментам варто пропонувати системи автоматичної класифікації текстових даних на наявність пропаганди, а працюючи із кожним сегментом, варто розробляти план впливу на нього окремо.

Сформуємо базову стратегію розвитку у цільових сегментах та наведемо її у таблиці 6.9.

Таблиця 6.9 — Визначення базової стратегії розвитку

Альтернатива розвитку продукту	Стратегія охоплення ринку	Конкурентноспроможні позиції відповідно до альтернативи	Базова стратегія розвитку
Стратегія спеціалізації	Пропонування продукту потенційним клієнтам, можлива модифікація ПО під потреби конкретного споживача	Довготривалі стосунки із клієнтами	Стратегія диференціації

У якості базової стратегії розвитку оберемо стратегію диференціації — стратегію, у якій орієнтування йде на потреби користувача. У разі провалу

такої ідеї, стратегію можна змінити на стратегію спеціалізації — тобто орієнтування на конкретну цільову групу.

Наступним кроком визначимо стратегію конкурентної поведінки на ринку, та наведемо її у таблиці 6.10.

Таблиця 6.10 — Стратегія конкурентної поведінки на ринку

Чи є проект “першопрохідцем” на цільовому ринку	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
Так	Шукати нових споживачів	Так як продукт є “першопрохідцем” на ринку, всі характеристики товару будуть створюватися з нуля під потреби користувача.	Стратегія заняття конкурентної ніші

Отже, обраною стратегією є зайняття конкурентної ніші з охопленням декількох сегментів споживачів, а оскільки продукт не є першопрохідцем на

ринку, то споживачів потрібно у більшості випадків забирати у існуючих конкурентів.

Визначена стратегія позиціонування продукту на ринку наведена у таблиці 6.11.

Таблиця 6.11 — Стратегія позиціонування продукту

Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту
Зменшення часу обробки тексту алгоритмом; Виокремлення окремих фраз, які є потенційно маніпулятивними; Робота з текстами не лише англійською, а і українською мовою;	Диференціації	Висока якість, налаштування на довготривалі стосунки із клієнтами	Швидкодія Клієнторієнтованість Точність класифікації

Висновок до розділу

Метою цього розділу є формування ідеї для розробки стартап проекту. Спочатку було проаналізовано існуючий аналог системи автоматичної класифікації текстових даних на наявність пропаганди та проведений порівняльний аналіз з розроблювальним продуктом. Єдиний аналог має схожий функціонал, але було визначено переваги розроблювального алгоритму перед вже існуючим.

Була проведена експертиза можливості втілення проекту з технічної точки зору, під час якої не було виявлено жодного ризику, оскільки для розробки треба використати ті інструменти, які вже існують та є безкоштовними.

Також був побудований маркетинговий план та визначені стратегії розвитку продукту на ринку.

ВИСНОВКИ

В процесі виконання даної магістерської дисертації було детально вивчено та ретельно розібрано існуючі методи машинного навчання для класифікації даних, результати їх застосування на зібраних вибірках даних. Було проаналізовано методи обробки неструкторованих тестових потоків даних, включаючи їх особливості. Розглянуто інструменти для обробки потоків тестових даних, наведено позитивні та негативні сторони кожного з них і обранано найбільш підходящий для виконання поставленого завдання.

Розглянута математична модель потоку текстових даних, процес обробки тестових повідомлень з потоку, використовуючи розроблений класифікатор. Зібрано тестовий набір даних, який складається з близько 4 тисяч тестових повідомлень з різних соціальних мереж. Описано математичну модель алгоритму побудови класифікатора – Gradient Boosting Tree. Також визначено та описано метрики визначення характеристик класифікатора.

В процесі виконання поставленого завдання розроблено e2e систему, для визначення сентименту повідомлень з месенджера телеграм, з його візуалізацією. Система складається з 4-х окремим підсистем, які розгортаються в окремий Docker-кластер, кожен з яких ізольований і може бути замінений на будь-яку іншу реалізацію.

В процесі експериментально було порівняно два алгоритми класифікації даних на різних вхідних конфігурації, а саме Random Forest, та Gradient Boosting Tree. Точність виконання Random Forest, на найкращій конфігураціїї – кількість дерев – 10, складає близько 81.1% точності на тестовому наборі та 71.37% на валідаційному. Що доволі не поганий показник. Для алгоритму ж Gradient Boosting Tree, найкраща конфігурація – глибина дерева – 6, та кількість ітерацій – 30. А точність складає 84.83% на тестовому, 74.01%

точності на валідаційному. Gradient Boosting Tree, на зібраному наборі даних працює точніше, тому в дослідеженні використовується саме він.

Наукова новизна отриманих результатів полягає в розробці та реалізації класифікатора сентименту в неструктурованих потоках даних для української мови, а також його візуалізація результатів.

Всі поставленні завдання даної роботи були виконанні і була досягнута поставлена мета. Було зібрано набір даних, та на основі його було навчено класифікатор для визначення сентименту в тектових повідомленнях українською мовою.

Список використаної літератури

1) Silvana C. Exploratory analysis of textual data streams / C. Silvana, F. Alfio, S. Stefano. // Future Generation Computer Systems. – 2017. – №68. – С. 391–406.

2) Четверкин И. И. Тестирование систем анализа тональности на семинаре РОМИП-2012 / И. И. Четверкин, Н. В. Лукашевич., 2013. – (РГГУ).

3) Попко А. В. Методи виявлення образ в коротких текстах : дис. канд. техн. наук : 151 / Попко Андрій Валентинович – Київ, 2018. – 133 с.

4) Шингалов Д. В. Методи автоматичного аналізу настроїв в соціальних мережах / Д. В. Шингалов. // Інформаційна безпека та комп'ютерні технології. – 2017. – С. 177–179.

5) Peter Turney. Semantic Orientation Applied to Unsupervised Classification of Reviews / Peter Turney. – Munich: Periodica Polytechnica Ser. Soc. Man. Scl., 2002. – 513 p.

6) Bo Pang. Opinion Mining and Sentiment Analysis. Foundations and Trends in Information Retrieval / Bo Pang, Lillian Lee // Foundations and Trends in Information Retrieval — 2008. — No. 2. — pp. 1-135.

7) Benjamin Snyder. Multiple Aspect Ranking using the Good Grief Algorithm / Benjamin Snyder, Regina Barzilay. – Hamburg: Morgan Kaufmann Publishers is an imprint of Elsevier., 2007. — 420 p.

8) Detecting sadness in 140 characters: Sentiment analysis of mourning michael jackson on twitter / E. Kim [та ін.] // Web Ecology. — 2009. — Т. 3. — С. 1— 15.

9) Github - Набір даних [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://github.com/dmytro-verner/sentiment-analysis-ukrainian-tweets.

10) Рисін А. Словник ВЕСУМ та інші пов’язані засоби NLP для української мови [Електронний ресурс] / А. Рисін, В. Старко, Д. Чаплинський // Російсько-українські словники. – 2017. – Режим доступу до ресурсу: https://r2u.org.ua/articles/vesum.

11) Методи автоматичного аналізу тональності контенту у соціальних мережах для виявлення інформаційно-психологічних впливів / Д. В.Шингалов, Є. В. Мелешко, Р. М. Минайленко, В. А. Резніченко. // Machinery in agricultural production, industry machine building, automation. – 2017. – №30. – С. 196–202.

12) Apache Storm [Електронний ресурс] // 2019 – Режим доступу до ресурсу: https://storm.apache.org/.

13) Apache Hadoop [Електронний ресурс] – Режим доступу до ресурсу: https://hadoop.apache.org/.

14) Splunk [Електронний ресурс] – Режим доступу до ресурсу: https://www.splunk.com/.

15) Amazon Kinesis [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/ru/kinesis/.

16) Apache Spark [Електронний ресурс] – Режим доступу до ресурсу: https://spark.apache.org/.

17) Apache Hive [Електронний ресурс] – Режим доступу до ресурсу: https://hive.apache.org/.

18) Apache Flink [Електронний ресурс] – Режим доступу до ресурсу: https://flink.apache.org/.

19) Sentiment analysis in Twitter / E. Martinez-C´amara [и др.] // Natural Language Engineering. — 2014. — Т. 20, № 01. — С. 1—28.

20) Extracting Verb Expressions Implying Negative Opinions. / H. Li [та ін.] // AAAI. — 2015. — С. 2411—2417.

21) *De Saeger S.*, *Torisawa K.*, *Kazama J.* Looking for trouble // Proceedings of the 22nd International Conference on Computational Linguistics-Volume

22) Ханько Г.В. Огляд та аналіз алгоритмів Text Mining [Текст]: / Г. Ханько, О.Гавриленко, Ю. Олійник //Матеріали VII Міжнародна науково-технічна конференція “Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління”, 2018 - Полтава, с. 120-124

23) Olena Gavrilenko, Yuri Oliynik, Hanna Khanko / Comparative analysis of text mining algorithms for identifying agitation data” / World Congress “Aviation in 21st century”, 2.3.80-2.3.83

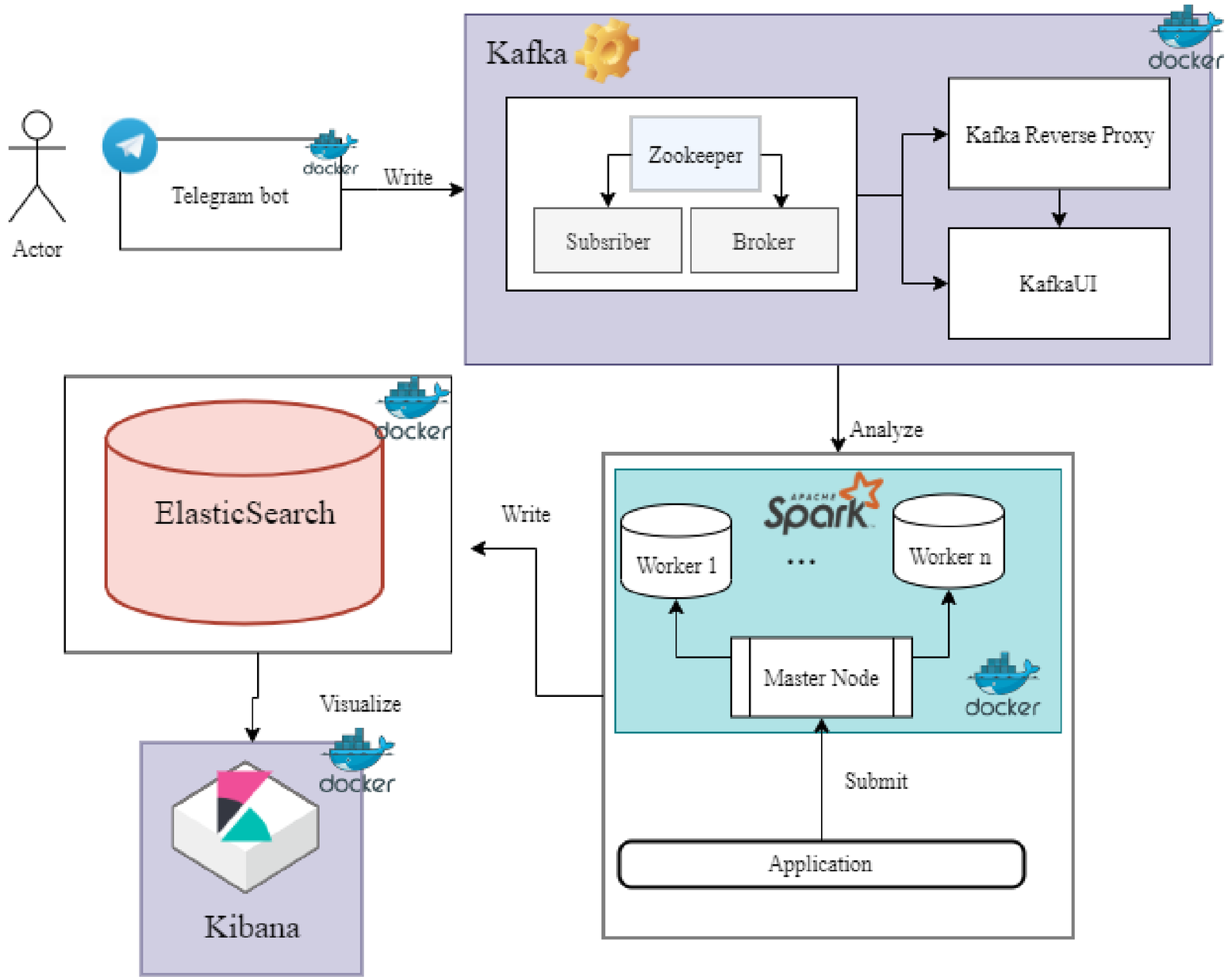
24) Egozzi O. Concept-based indexing text information retrieval [Текст]: / O. Egozi, S. Markovitch, E. Gabrilovich // ACM Transactions on Information System (TOIS). 2011, vol.29 #2. pp. 4-6.

25) Boubekeur F. Concept-based indexing in text information retrieval [Текст]: / F. Boubekeur, W. Azzoug // Int. J. Comput. Sci. Inf. Technol. 2013, vol. 5 pp. 119- 136.

26) Huang L. Learning a concept-based document similarity measure. [Текст]: / L. Huang // Journ. of the American Society for Information Science and Technology. 2012, vol. 63, #8. pp.1593-1608.

Додаток А
Графічний матеріал

СХЕМА ОБРОБКИ ПОТОКІВ ДАНИХ



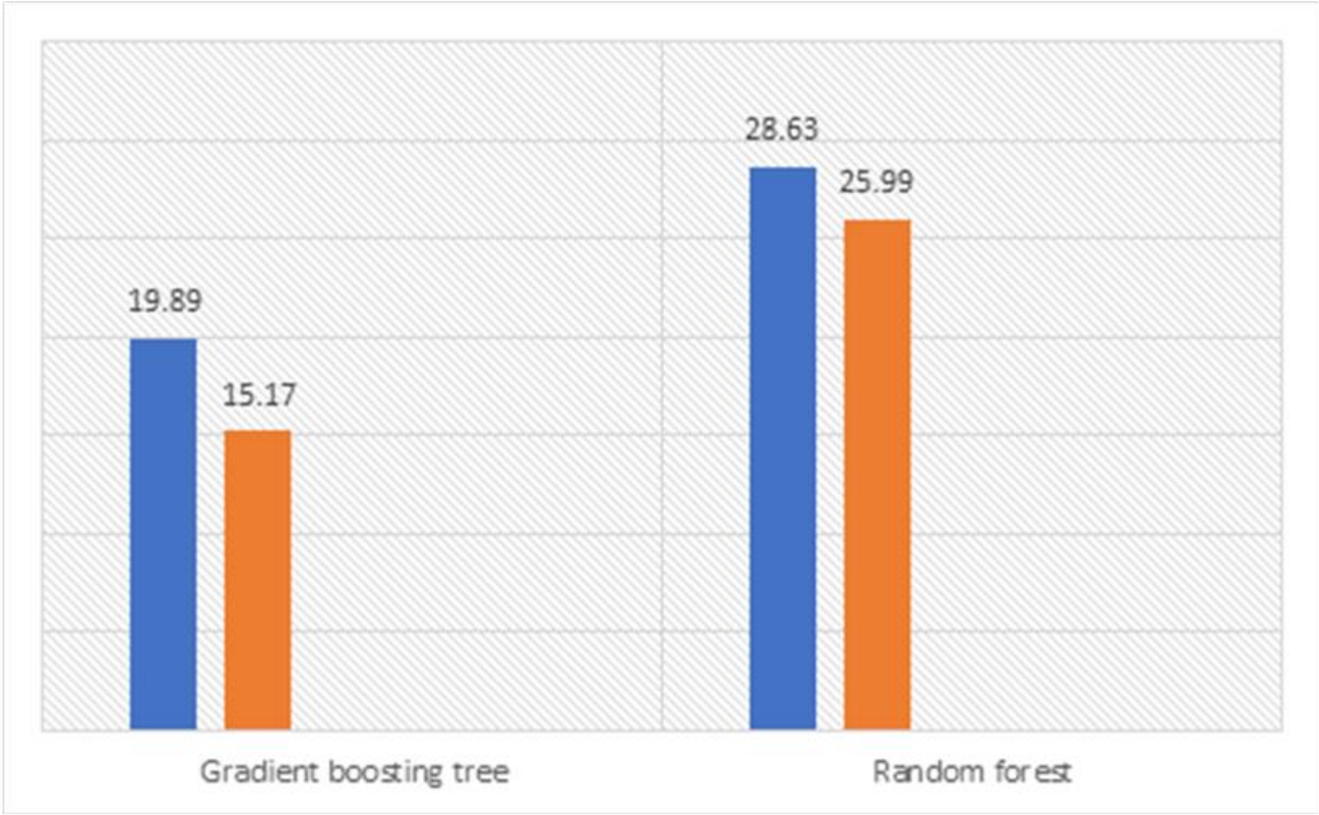
Демонстраційний плакат до магістерської дисертації

Схема обробки потоків данних

Виконав студент гр. ІП-82мп Степанюк Є.Ю.

Керівник Олійник Ю.О.

ПОРІВНЯННЯ РЕЗУЛЬТАТІВ ТЕСТУВАННЯ АЛГОРИТМІВ RANDOM FOREST ТА GRADIENT BOOSTING TREE



Демонстраційний плакат до магістерської дисертації

Порівняння результатів тестування алгоритмів random forest та gradient boosting tree

Виконав студент гр. ІП-82мп Степанюк Є.Ю.

Керівник Олійник Ю.О.

Додаток Б

Лістинг коду

```
package com.yevhenii


import java.time.format.DateTimeFormatter


import com.cybozu.labs.langdetect.DetectorFactory

import org.apache.spark.{SparkConf, SparkContext, mllib}

import org.apache.spark.sql.SparkSession


import scala.util.Properties

import org.apache.spark.sql._

import org.apache.spark.sql.types._

import org.apache.spark.ml.classification.GBTClassifier

import org.apache.spark.sql.SparkSession

import org.apache.spark.sql.functions.{col, from_json, split}

import org.apache.log4j.{Level, Logger}

import org.apache.spark.mllib.feature.HashingTF

import org.apache.spark.mllib.regression.LabeledPoint

import org.apache.spark.mllib.tree.GradientBoostedTrees

import org.apache.spark.mllib.tree.configuration.BoostingStrategy

import org.apache.spark.mllib.tree.model.GradientBoostedTreesModel

import scala.util.Try

import org.apache.spark.ml.linalg

import org.apache.spark.sql.Row

import org.apache.spark.rdd.RDD

import org.apache.spark.sql.Row

import org.apache.spark.ml.linalg.Vectors

import org.elasticsearch.spark.sql._

import org.elasticsearch.spark._

import org.elasticsearch.hadoop.cfg.ConfigurationOptions


import scala.reflect.macros.whitebox


object Application {

 val positiveLabelWord = "добре"

 val negativeFirstLabelWord = "погано"

 val negativeSecondLabelWord = "поганий"


 val hashingTF: HashingTF = new HashingTF(2000)


 def main(args: Array[String]): Unit ={


Logger.getLogger("org").setLevel(Level.ERROR)


   val kafkaAddress = Properties.envOrElse("KAFKA_ADDRESS", "")

   val kafkaTopic = Properties.envOrElse("KAFKA_TOPIC", "")

   var esUrl = Properties.envOrElse("ELASTICSEARCH_URL", "elasticsearch:9200")

   var esIndex = Properties.envOrElse("ELASTICSEARCH_INDEX", "telegram/es_spark")


   val appName = "Application"

   val conf = new SparkConf()


conf.setAppName(appName).setMaster("local")

     .set("spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-10:2.4.0").set("spark.driver.allowMultipleContexts", "true")


   conf.set("es.index.auto.create", "true").set("es.nodes", esUrl)


   val model: GradientBoostedTreesModel = train(conf)


   val spark: SparkSession = SparkSession.builder.config(conf).getOrCreate();

   import spark.implicits._


DetectorFactory.loadProfile("src/main/resources/profiles")


   val schema = new StructType()

     .add("message",StringType)

.add("user_full_name",StringType)

.add("from_username",StringType)

     .add("date",StringType)
```

```
    .add("chat_title",StringType)
    .add("chat_name",StringType)
    .add("chat_id",StringType)

  val df = spark
    .readStream
    .format("kafka")

.option("kafka.bootstrap.servers",
kafkaAddress)
    .option("subscribe", kafkaTopic)
    .load()
    .select(from_json(col("value")
     .cast("string"), schema)
     .as("data"))
    .select("data.*")
    .map { row =>
     Message(
       row.getString(0),
       row.getString(1),
       row.getString(2),
       row.getString(3),
       row.getString(4),
       row.getString(5),
       row.getString(6),

detectLanguage(row.getString(0)),

model.predict(toLabeled(row.getStri
ng(0)).features))
    }

  df.writeStream
    .format("console")
    .outputMode("append")
    .start()
      .awaitTermination()
  }

 def convertToVector(str: String):
mllib.linalg.Vector = {
   return
org.apache.spark.mllib.linalg.Vector
s.dense(str.split(" ").map(s =>
s.toDouble))
  }
 def train(conf: SparkConf):
GradientBoostedTreesModel ={

   val startTime =
System.nanoTime()

   val sc = new SparkContext(conf)
   val sqlContext = new
org.apache.spark.sql.SQLContext(sc
)

   val tweetDF =
sqlContext.read.json("src/main/reso
urces/dataset/dataset.json")

   val messages =
tweetDF.select("message",
"isPositive")
   println("Total messages: " +
messages.count())

   val positiveMessages =
messages.filter(messages("isPositiv
e").contains(true))
   val countPositive =
positiveMessages.count()

   val negativeMessages =
messages.filter(messages("isPositiv
e").contains(false))
   val countNegative =
negativeMessages.count()

   println("Number of positive
messages: " + countPositive)
   println("Number of negative
messages: " + countNegative)

   val smallestCommonCount =
Math.min(countPositive,
countNegative).toInt

   val messagesWorkingSet =
positiveMessages.limit(smallestCo
mmonCount).unionAll(negativeMes
sages.limit(smallestCommonCount)
)

   val messagesRDD =
messagesWorkingSet.rdd
   //filter out tweets that can't be
parsed
   val labeledTweets =
getLabeledTweets(messagesRDD)

   //Map the input strings to a tuple
of labeled point and input text
   val inputLabeled =
labeledTweets.map(
    t => (t._1,
hashingTF.transform(t._2)))
    .map(x => new
LabeledPoint(x._1.toDouble, x._2))

inputLabeled.take(5).foreach(println
)

   val sampleSet =
labeledTweets.take(1000).map(
    t => (t._1,
hashingTF.transform(t._2), t._2))
```

```
    .map(x => (new LabeledPoint(x._1.toDouble, x._2), x._3))


   // separate the data into two sets (30% held out for validation testing)
   val splits = inputLabeled.randomSplit(Array(0.7, 0.3))
   val (trainingData, validationData) = (splits(0), splits(1))


   val boostingStrategy = BoostingStrategy.defaultParams("Classification")

boostingStrategy.setNumIterations(30)

boostingStrategy.treeStrategy.setNumClasses(2)

boostingStrategy.treeStrategy.setMaxDepth(6)


   val model: GradientBoostedTreesModel = GradientBoostedTrees.train(trainingData, boostingStrategy)
   // evaluate model on test instances and compute test error
   val labelAndClassTrainingSet = trainingData.map { point =>
    val prediction = model.predict(point.features)
     Tuple2(point.label, prediction)
   }


   val labelAndClassValidationSet = validationData.map { point =>
    val prediction = model.predict(point.features)
     Tuple2(point.label, prediction)
   }


   val results = labelAndClassTrainingSet.collect()


   var positiveTotal = 0
   var positiveCorrect = 0
   var negativeTotal = 0
   var negativeCorrect = 0
   results.foreach(
    r => {
     if (r._1 == 1) {
      positiveTotal += 1
     } else if (r._1 == 0) {
      negativeTotal += 1
     }
     if (r._1 == 1 && r._2 ==1) {
      positiveCorrect += 1
     } else if (r._1 == 0 && r._2 == 0) {
      negativeCorrect += 1
     }
    }
   )


   //calculate test error
   val testErrorTrainingSet = labelAndClassTrainingSet.filter(r => r._1 != r._2).count.toDouble / trainingData.count()
   //pull up the results for validation set
   val validSetResults = labelAndClassValidationSet.collect()


   var positiveTotalValidSet = 0
   var negativeTotalValidSet = 0
   var positiveCorrectValidSet = 0
   var negativeCorrectValidSet = 0
   validSetResults.foreach(
    r => {
     if (r._1 == 1) {
      positiveTotalValidSet += 1
     } else if (r._1 == 0) {
      negativeTotalValidSet += 1
     }
     if (r._1 == 1 && r._2 ==1) {
      positiveCorrectValidSet += 1
     } else if (r._1 == 0 && r._2 == 0) {
      negativeCorrectValidSet += 1
     }
    }
   )


   val testErrorValidationSet = labelAndClassValidationSet.filter(r => r._1 != r._2).count.toDouble / validationData.count()


   val predictions = sampleSet.map {
    point =>
     val classifiedValue = model.predict(point._1.features)
     (point._1.label, classifiedValue, point._2)
   }


   //the first value is the real class label. 1 is positive, 0 is negative.
   //class is the second value
   predictions.take(100).foreach(x => println("label: " + x._1 + " class:
```

```
" + x._2 + " text: " + x._3.mkString(" ")))


   val endTime = System.nanoTime()


   println("negative messages in Training Set: " + negativeTotal + " positive messages: " + positiveTotal)

   println("positive % correct: " + positiveCorrect.toDouble/positiveTotal)

   println("negative % correct: " + negativeCorrect.toDouble/negativeTotal)

   println("Test Error Training Set: " + testErrorTrainingSet)


   println("negative messages in Validation Set: " + negativeTotalValidSet + " positive messages: " + positiveTotalValidSet)

   println("positive % correct: " + positiveCorrectValidSet.toDouble/positiveTotalValidSet)

   println("negative % correct: " + negativeCorrectValidSet.toDouble/negativeTotalValidSet)

   println("Test Error Validation Set: " + testErrorValidationSet)


   println("Elapsed time: " + (endTime - startTime) / 1E9 + "secs")


   return model

  }


  def detectLanguage(text: String) : String = {

   Try {

      val detector = DetectorFactory.create()

      detector.append(text)

      detector.detect()

    }.getOrElse("unknown")


  }


  def toLabeled(msg: String): LabeledPoint = {

   val messageSanitized = msg.toLowerCase().replaceAll(positiveLabelWord, "")


.replaceAll(negativeFirstLabelWord, "")


.replaceAll(negativeSecondLabelWord, "")


   val msgSeq = (1, messageSanitized.split(" ").toSeq)

   val t = (msgSeq._1, hashingTF.transform(msgSeq._2))


   return new LabeledPoint(msgSeq._1.toDouble, hashingTF.transform(msgSeq._2))

  }


  def getLabeledTweets(messagesRDD: RDD[Row]): RDD[(Int, Seq[String])] ={

   //filter out tweets that can't be parsed

   val positiveAndNegativeRecords = messagesRDD.map(

      row =>{

        Try{

         val msg = row(0).toString.toLowerCase()

         val isPositiveStatus = row(1).toString.toLowerCase()

         var isPositiveLabel = 0

         //filter by two negative words

         if(isPositiveStatus == "true"){

           isPositiveLabel = 1

         }else if(isPositiveStatus == "false"){

           isPositiveLabel = 0

         }

         val messageSanitized = msg.toLowerCase().replaceAll(positiveLabelWord, "")


.replaceAll(negativeFirstLabelWord, "")


.replaceAll(negativeSecondLabelWord, "")


         (isPositiveLabel, messageSanitized.split(" ").toSeq) //tuple returned

        }

      }

    )


   //filter out exceptions

   val exceptions = positiveAndNegativeRecords.filter(_.isFailure)

   println("Total records with exceptions: " + exceptions.count())

   exceptions.take(10).foreach(x => println(x.failed))


   val labeledTweets = positiveAndNegativeRecords.filter(_.isSuccess).map(_.get)
```